ADVANCE THE DNA COMPUTING

A Dissertation

by

ZHIQUAN FRANK QIU

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 2003

Major Subject: Computer Engineering

ADVANCE THE DNA COMPUTING

A Dissertation

by

ZHIQUAN FRANK QIU

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Approved as to style and content by:

Mi Lu (Chair of Committee)	
M. Ray Mercer (Member)	Gwan Choi (Member)
Don Friesen (Member)	Chanan Singh (Head of Department)

August 2003

Major Subject: Computer Engineering

ABSTRACT

Advance the DNA Computing. (August 2003)

Zhiquan Frank Qiu ,

B.S., University of Electronic Science and Technology of China;

M.S., Virginia Polytechnic Institute and State University

Chair of Advisory Committee: Dr. Mi Lu

It has been previously shown that DNA computing can solve those problems currently intractable on even the fastest electronic computers. The algorithm design for DNA computing, however, is not straightforward. A strong background in both the DNA molecule and computer engineering are required to develop efficient DNA computing algorithms. After Adleman solved the Hamilton Path Problem using a combinatorial molecular method, many other hard computational problems were investigated with the proposed DNA computer. The existing models from which a few DNA computing algorithms have been developed are not sufficiently powerful and robust, however, to attract potential users.

This thesis has described research performed to build a new DNA computing model based on various new algorithms developed to solve the 3-Coloring problem. These new algorithms are presented as vehicles for demonstrating the advantages of the new model, and they can be expanded to solve other NP-complete problems. These new algorithms can significantly speed up computation and therefore achieve a consistently better time performance. With the given resource, these algorithms can also solve problems of a much greater size, especially as compared to existing DNA computation algorithms. The error rate can also be greatly reduced by applying these

new algorithms. Furthermore, they have the advantage of dynamic updating, so an answer can be changed based on modifications made to the initial condition. This new model makes use of the huge possible memory by generating a "lookup table" during the implementation of the algorithms. If the initial condition changes, the answer changes accordingly. In addition, the new model has the advantage of decoding all the strands in the final pool both quickly and efficiently. The advantages provided by the new model make DNA computing an efficient and attractive means of solving computationally intense problems.

ACKNOWLEDGMENTS

I wish to express my sincerest thanks to my advisor, Professor Mi Lu, the chairman of the committee, for her guidance, patience, and support throughout the course of my Ph.D. studies. I have learned much as a direct result of working with her. My thanks go to Dr. M. Ray Mercer for his support and his confidence in my abilities, and for his advices for my career. I also thank Dr. Gwan Choi and Dr. Don Friesen for their helpful advice and for being on my committee.

I dedicate this thesis to my father, Zhongze Qiu, and my mother, Xingmei Liu, for their unflagging faith in me. Without their devotion, support, patience, and encouragement throughout the period of study, this work would not have been possible. Special thanks to my brother and sister in law for their love and encouragement. Finally, I show my special appreciation to my wife, Jiali Chen for her love to me.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

CHAPTER I

INTRODUCTION

A. Motivation

Human beings began calculating somewhere around 2000 BC. Driven by necessity, chance, and inventiveness, we advanced from using our fingers to marking on tablets, from the abacus to the mechanical adding machine, and finally to the electronic computer. The modern computer has had a significant impact on everyday life, giving us the ability to compute, letting us create complex algorithms to quickly solve problems, sometimes in a matter of milliseconds [1].

The computer's influence on people's lives has been well documented. Its impact on the economy is also commonly known. The current trend of steady growth and accelerating productivity rely heavily on continued improvements in computer productivity [2]. In 1999, US companies spent over $200 billion on computers and related items, more than they invested in any other type of capital good [3].

Nevertheless, today's computers have their limitations. The transistors on a silicon chip have been doubling in number roughly every 18 to 24 months. The shrinking device size and increasing density that prompts this doubling will cause physical problems in the coming decades, even though they currently provide increase in speed and funcationality with a substantial drop in costs. Problems with the small size of the devices (about 90 nanometers (nm)) that make up the electronic computer architectures include the eventual electron leakage across a small number of atoms, a lack of uniformity in the distribution of dopants in semiconductors, a low yield in the number of usable chips manufactured, and heat dissipation from the high density

---

The journal model is *IEEE Transactions on Automatic Control.*

of devices on the chip. Furthermore, as chip manufacturing becomes more and more complex, the current $1 billion IC-manufacturing facilities will make the facbrications no longer economically feasible. If we want to keep up the pace of computer cost and performance improvements in the long term, researchers will have to fully explore these issues and possibly explore alternative technologies [1].

Computing systems inspired by biological systems (biocomputation) offer one possible alternative that is currently under investigation. DNA carries the genetic information for life as we know it. Before its identification by Watson and Crick in 1953, the quantum physicist Schrodinger had already accurately predicted the carrier of genetic information to be an "aperiodic crystal": a structured medium (crystal) capable of storing information because of variations allowed within the medium's structure (aperiodicity) [4].

A single strand of DNA is a string consisting of a combination of four different base nucleotides: A(adenine), C(cytosine), G(guanine) and T(thymine). When attached to deoxyribose, these base nucleotides can be strung together to generate long sequences. Each single string can be paired up with a complementary string to form a double helix. This pair-up only occurs under the WC(Watson-Crick) complement rule. That is, A only pairs with T and G only pairs with C. Also the double strand can be separated by heating. The dissociated strands separate from each other without breaking the chemical bonds that hold the nucleotides together along the single strand. Either one of these denatured single strands or both together can be used for further operations because they perfectly complement one another. One good example of this idea is the PCR (polymerase chain reaction) method. It is used to initialize test tubes in many DNA computing algorithms by making numerous identical strands through a repetition of the above procedure.

Since Watson and Crick's discovery, many ways to manipulate DNA have been

developed. Biological techniques include the use of enzymes for cutting and pasting, and the polymerase chain reaction for the reproduction of DNA strands. Biotechnological techniques include the selective filter, tagging and DNA sequencing. Together, these devlopments enable us to use DNA as a modifiable storage medium-a kind of memory. These developments also allow us to use these techniques as operations on that memory in order to implement algorithms.

DNA was first used for computations in 1994. In his ground breaking Science article, Leonard Adleman described his experiment that solved a seven-node instance of the Hamiltonian path problem from graph theory. (The problem involves finding a path containing all nodes only once, through a mathematical graph.) He devised a code for the edges of a graph based on the encodings of their nodes.

As a result, Adleman produced sequences that corresponded to candidate solution paths by randomly gluing together sequences of single nodes. By producing enough of these sequences, all candidate solutions were constructed. This construction was done in parallel. In other words, all strands underwent the reactions simultaneously.

Through a sequence of filtering steps, strands that were of the wrong length or that did not contain all the required nodes were eliminated. Only those strands that corresponded to actual solutions were kept. This filtering was also done concurrently on all strands. The fact that DNA strands remained after this process indicated that solutions to the problem existed; by sequencing the remaining DNA, a single solution was decoded.

Since Adleman [5] and Lipton presented the idea of solving difficult combinatorial search problems using DNA molecules, there has been some new work regarding how DNA could be used for computations [6] [7] [8] [9] [10] [11] [12] [13]. Since one liter of water can hold 10^{22} bases of DNA, these methods all take advantage of the massive

parallelism made available by DNA computers. This capability also raises the hope of solving those problems currently intractable for electronic computers.

However, most potential users are still watching DNA computing develop without taking any action of their own. This is primarily because the completion of each biological operation can take a substantial amount of time. The implementation of each algorithm for a solved problem can take weeks, or even months. To make the situation even worse, the long process of developing the algorithms must be restarted if the initial condition changes. These obstacles must be overcome before any substantial progress in DNA computing will be accomplished. There are two primary tasks that must be accomplished. The first is the speedup of the algorithms. The second is finding a solution to a number of similar problems simultaneously or the real time updating of a solution.

B. Objective

The primary objective of this research is to advance DNA computing so that it can be made more attractive to potential users. These new methods will increase the potential of DNA computing. They will offer ways for DNA computing technology to be used to solve problems that currently are considered unsolvable. These new methods will also make DNA computing more cost and time efficient.

More specifically, the first objective of this research is to speed up the algorithms and to increase the size of the problems that can be solved. The existing algorithms currently take a long time to finish- months or even years. The new method will parallelize the processes of the algorithms so that the implementation of new algorithms can be accomplished much more quickly. The new method may even reduce the error rate.

The second objective of this research is to real time update a solution when the initial condition of a problem changes. After an initial answer is generated, the initial condition of the problem may change. This now results in a need to start the algorithm again no matter how small the initial change is. Instead, by real time updating the solution, new answers can be found simply by going through a few extra operations.

The third objective of this research is to find a way to discover all possible exact answers to a problem both quickly and efficiently. When DNA molecular strands are used to compute, a set of strands stays in the pool to represent the final answer. This new method can decode these strands efficiently and quickly.

C. General Assumptions and Limitations

While presenting our new algorithms (except the error resistant one), we made the assumption that all molecular biological procedures are error free. This is not true in reality, but there is a significant body of finished and ongoing research which attacks the problem of error-resistant implementations [14] [15] [16] [17] [18] [19]. This work has showed many fault tolerant techniques [20] [21] [22] and error-correction methods [17]. Good coding methods may also minimize the possible error rate [23] [24] [25] [26]. It is reasonable to assume that errors which arise during DNA computing operations can be dealt with through these techniques and the new techniques provided here.

D. Outline of the Dissertation

This dissertation consists of four chapters. Chapter II describes the current state of DNA computing. It also discusses the advantages offered by the new approaches described in this dissertation. Chapter III describes the new DNA computing model

and how new algorithms can be designed based on it. This chapter also provides these algorithms whose designs are based on the new model developed to improve the performance of DNA computing, especially in terms of speedup, realtime updating and quick decoding. Chapter IV summaries the contributions of this study and describes the future research necessary to make DNA computing more attractive.

CHAPTER II

BACKGROUND

Since Adleman's 1994 demonstration of the possibility of solving computationally intensive problems using DNA molecules [5], some DNA algorithms have been developed [27] [28] [29] [8] [30] [10] [31] that attack a number of DNA computing problems. Adleman et al. have attempted to design basic DNA computing operations and have also tried to build a DNA computer [5] [32] [33] [34] [35]. Winfree et al. have also been building different models for DNA computing [36] [37] [38] [39] [40] [29] [41]. Some error detection and fault tolerance methods have been found by Seeman et al. [16] [42] [18] [43] [44]. Condon et al. [15] [24] [45] and Smith et al. [22] [21] [26] have generated efficient coding methods for DNA computing and have made significant progress in accomplishing surface-based DNA computing. This work has accomplished much lower error rate as compared to the solution-based approach. DNA algorithms for simple boolean and arithmetic computing have also been developed [46] [27] [47] [48] [49] [50] [51] [13]. All of these extensive efforts seem to indicate a bright future for DNA computing. Nevertheless, DNA computing is still not available for use in real applications for the following reasons: it is a fairly slow process due to slow bio-reactions; there is significant expense due to the costliness of bio-operations; and there are some unavoidable errors. The progress of this research toward eliminating some of these burdens is explained next.

DNA computing uses DeoxyriboNucleic Acid (DNA) strands as the basic processing units. A DNA computer is basically a collection of specially selected DNA strands (and the set of biological processes to manipulate them) whose combinations will result in the solution to some problems. Each strand of DNA encodes the state of a processor. Each processor operates independently. Technology is currently avail-

able both to select the initial strands and to filter the final solution. The promise of DNA computing is the massive parallelism. One liter of water can hold 10^{22} bases of DNA strands and all strands can be processed at the same time [52]. The already developed algorithms currently take advantage of the massive processing power made available by DNA computing [53] [54] [55]. If each strand processes one data item through performance of bio-operations, 10^{22} different operations can be completed in one bio-cycle with the help of the 10^{22} strands in the one liter of water. This raises the hope of solving problems currently considered intractable on available electronic computers. The necessary operation would take a 1GHz electronic computer 10^6 years to complete. Much effort has been put into locating a "killer application" which would attract the industrial world to the new DNA computers [56] [49]. Not only could these DNA computing algorithms solve computationally intensive problems, they can also lay down a basis for fundamental arithmatic and logical operations [9] [47] [48] [51]. However, unlike those in the electronic computers, the algorithmic design of DNA computing is not straightforward. A strong background in both the DNA molecule and computer engineering is required to develop efficient algorithms for DNA computing.

Although DNA computing can solve computationally intensive problems, it still takes longer than a desirable time to execute an algorithm. Improvement in the performance of DNA computing is crucial before DNA computing can be made really attractive for prefessional use. There has been some effort made to analize the complexity of DNA computing algorithms [57], but little has been accomplished toward improving the performance of these algorithms [58].

A. The Status of Existing DNA Computing Models

There are several DNA computing models that have already been developed. Most existing DNA computing algorithms have been developed for these existing models. An analysis of these existing DNA computing models shows why these models cannot satisfy the requirements of potential users.

1. The Self-Assembly Based Model

DNA self-assembly is a method used to construct molecular scale structures. In this method, artificially synthesized single strand DNA self-assembles into DNA crossover molecules, or tiles. These DNA tiles have sticky ends that preferentially match the sticky ends of certain other DNA tiles, facilitating further assembly into tiling lattices. Figure 1 provides an example of the grammer of this self-assembly. It uses rules of form $A \rightarrow pB$ and $A \rightarrow p$ where A and B are non-terminal symbols and p is a string of terminals. A language generated by regular grammar is called a regular language. For example, consider the grammar $G_E = \{S \rightarrow 0S,\ S \rightarrow 1T,\ S \rightarrow 0,\ T \rightarrow 0T,\ T \rightarrow 1S,\ T \rightarrow 1\}$ where 0 and 1 are terminals. This grammar gives rise to all bit strings with an even number of 1's. 110011 is a good example because $S \rightarrow 1T \rightarrow 11S \rightarrow 110S \rightarrow 1100S \rightarrow 11001T \rightarrow 110011$. Note that during the derivation, there is always a single nonterminal to the right of where the actions take place.

In this case, self-assembly is the spontaneous self-ordering of substructures into superstructures driven by the annealing of Watson-Crick base-paring DNA sequences. Computation by self-assembly entails the building up of superstructures from starting units such that the assembly process itself performs actual computation. Adleman made use of a simple form of this kind of computation by using self-assembly in his original experiment [5]: instead of blindly generating all possible sequences of vertices,

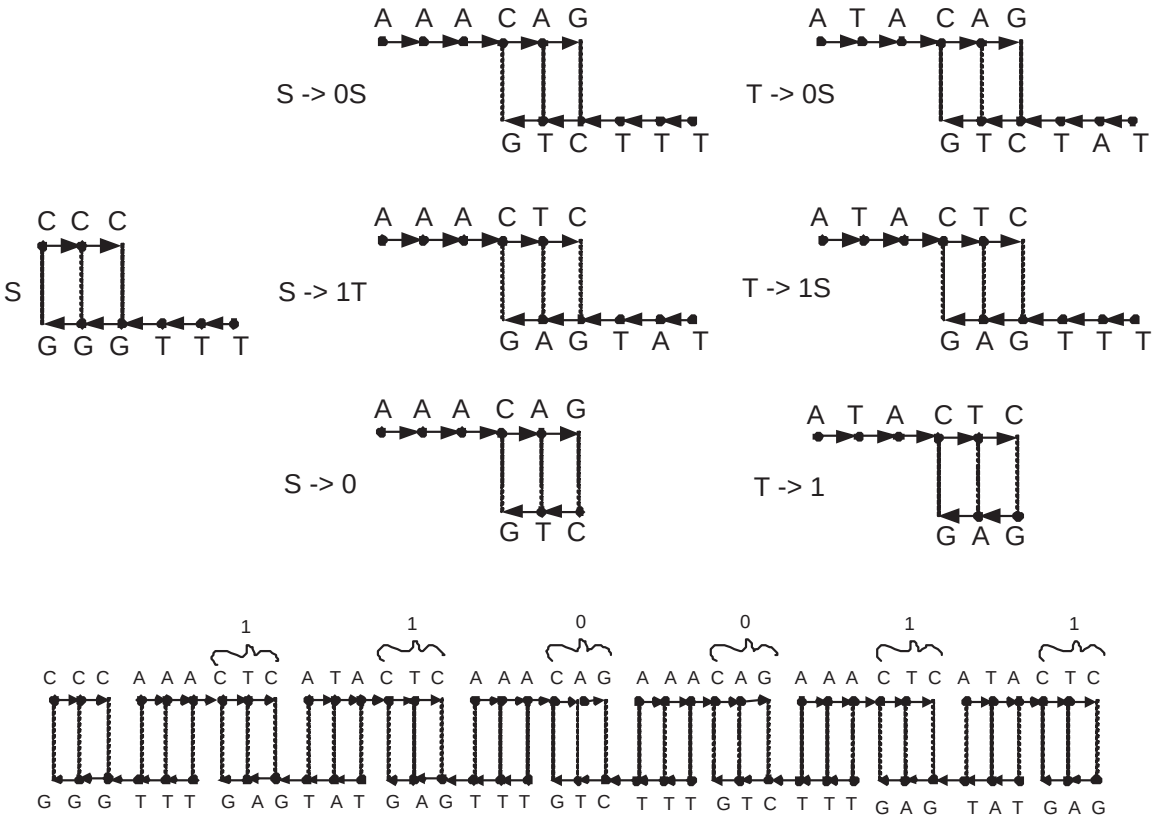


Fig. 1.: SELF ASSEMBLY TERMINALS AND PROCESSES

he instead used the oligonucleotide sequences and the logic of Watson-Crick to complementarily guide the self-assembly process to generate only valid paths. Winfree el al [37] generalized this approach for two-dimensional (2D) self-assembly processes and has shown that computation by self-assembly is Turing-universal.

Winfree et al and Eng [37][59] proposed the self-assembly of linear, hairpin, and branch DNA molecules in order to generate regular, bilinear, and context-free languages, respectively. They [37],[60] all proposed the use of self-assembled DNA nanostructures to solve NP complete combinatorial search problems.

The approach of programming the DNA self-assembly of tilings requires the following: (i) mixing the input of oligonucleotides to form the DNA tiles, (ii) allowing

the tiles to self-assemble into superstructures, and (iii) performing a single separation to identify the correct output.

The problem with the self-assembly model is that the algorithms are usually slow due to the low growth rate. When the temperature is raised to accelerate the process, the error rate increases exponentially and goes out of control [39].

2. Sticker Based Model

The sticker based model employs two basic groups of single strand DNA molecules in its representation of a bit string. Consider a *memory strand*, N bases in length, in K non-overlapping regions, each M bases long (thus $N \geq MK$). Each region is identified by exactly one bit position (or, equivalently, one boolean variable) during the course of the computation. K different *stickerstrands*, or simply *sticker*, are also designed. Each sticker is M bases long and is complementary to one, and only one of the K memory regions. If a sticker is annealed to its matching regions on a given memory strand, then the bit corresponding to that particular region is *on* for that strand. If no sticker is annealed to a region, then that region's bit is off. Figure 2 illustrates this representation scheme [34] [35] [32].

Each memory strand, along with its annealed stickers (if any), represents one bit string. Such partial duplexes are called *memory complexes*. A large set of bit strings is represented by a large number of *identical* memory strands, each of which has a sticker annealed only at the required bit positions. Such a collection of memory complexes is called a *tube*.

The four principle operations include: the *combination* of two sets of strings into one new set, the *separation* of one set of strings into two new sets, and the *setting* or *clearing* of the k^{th} bit of every string into a set. Figure 3 summarizes these required DNA interactions. The corresponding interpretation in terms of the DNA

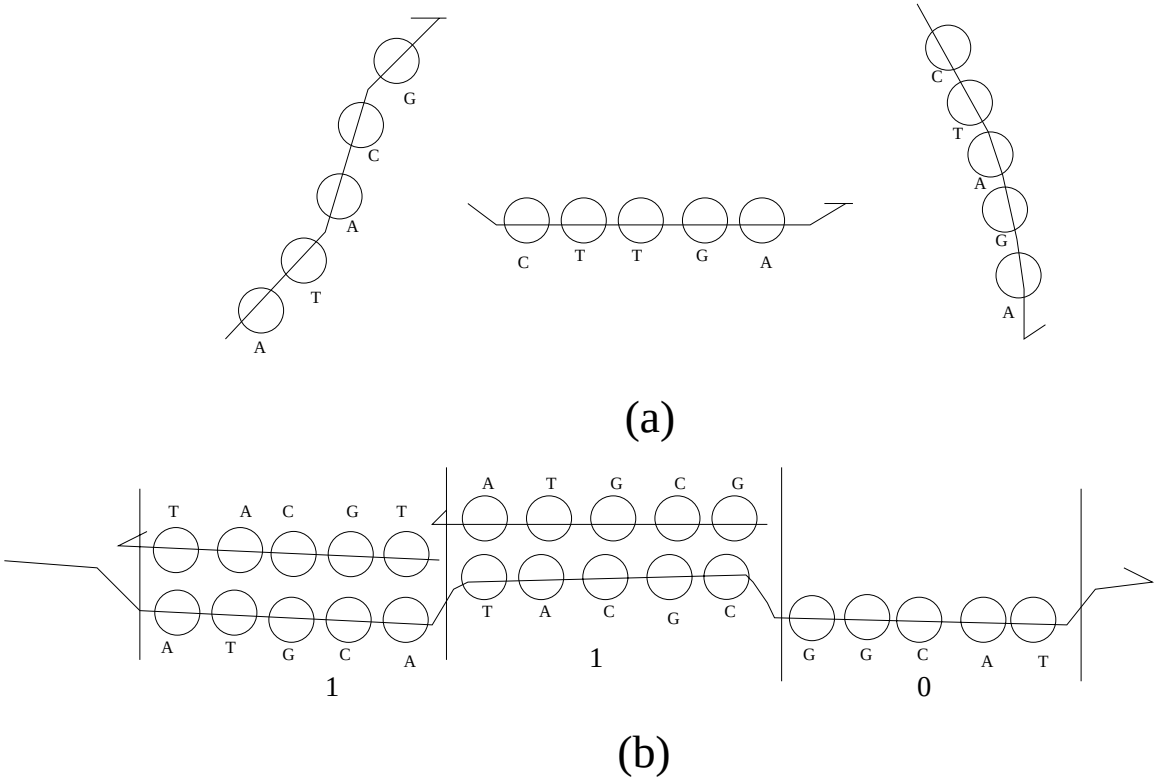


Fig. 2.: STICKERS AND SWITCHES (A) STICKERS (B) TWO ANNEALED STICKERS FOR TWO BITS ON

representation of these operations is listed below:

The most basic operation is to *combine* two sets of bit strings into one. This produces a new set containing the multi-set union of all strings from two input sets. In DNA, this corresponds to producing a new tube containing all of the possible memory complexes (with their annealed stickers undisturbed) from both input tubes.

A set of strings may be *separated* into two new sets, one containing all of the original strings that have a particular bit *on*, and the other containing all of those with the bit off. This corresponds to isolating from the set's tube only those complexes with a sticker annealed to the given bit's region. The original input set or tube is then destroyed.

To *set* (turn on) a particular bit in every string in a set, the sticker for that bit is annealed to the appropriate region on every structure in the set's tube (or left in place if already annealed).

To *clear* (turn off) a bit in every string of a set, the sticker for that bit must be removed (if present) from every memory complex in that set's tube.

The advantage of this model is that initialization is simple. All strands in the initial set are exactly the same. Synthesis of the initial solution space is quick and cheap when using the standard technology.

The problem with this sticker based model is that those stickers that annealed to the long strand may fall off during the process. The most difficult problem with this model is the *clear* operation. It requires removing the stickers for only that bit from every structure in the tube. Simple heating will obviously not work since all stickers from all bit regions would come off. Roweis et al. has recommend the

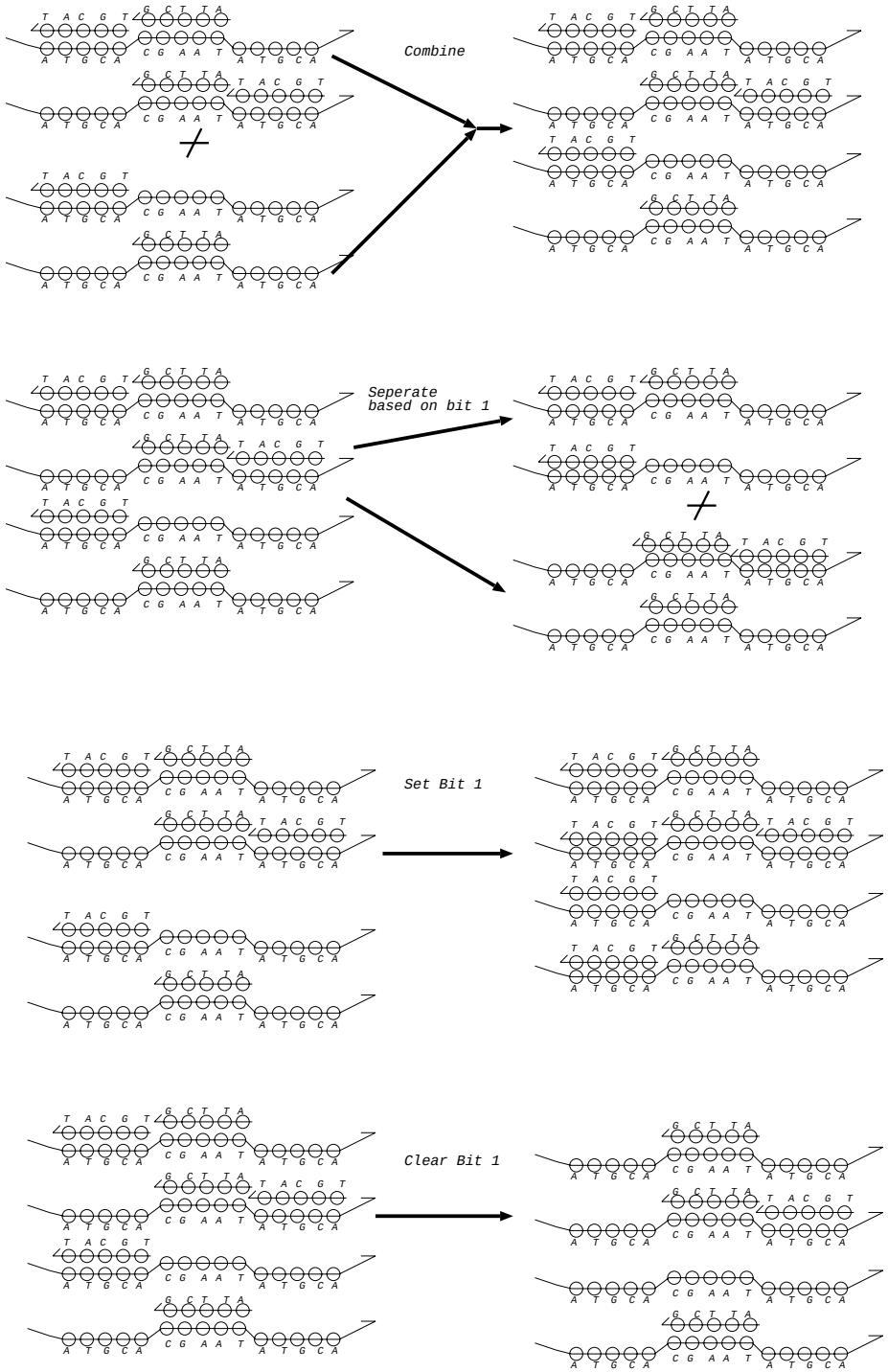


Fig. 3.: DNA MANIPULATIONS REQUIRED FOR THE FOUR OPERATIONS OF THE STICKER MODEL

possibility of designating certain bit regions as *weak* regions. These regions have weak stickers which dissociate more easily from the memory strand than regular stickers. By heating to some intermediate temperature, the *weak* stickers could be made to simultaneously dissociate, keeping all of the regular stickers in place [34]. This idea has increased the flexibility of DNA computing considerably.

3. Surface Based Model

The surface based model was first introduced by Liu et al. [22]. This surface based DNA computing methodology immobilizes the DNA strands on a particular surface (glass, silicon, gold, or beads). The strands are then subjected to operations such as hybridization from a solution or exonuclease degradation in order to extract the desired solution. This method greatly reduces the loss of DNA molecules during the purification steps. It is well known that surface based chemistries have become the standard for complex chemical syntheses such as solid-phase DNA synthesis, solid-phase protein sequence analysis, and many other chemistries [22].

After the initial solution space is defined as the set S of binary strings of length n, the following operations may be performed on S.

mark(i, b) : this marks all strings of S in which the ith bit has the value b.

mark$(i_1, b_1), (i_2, b_2), \ldots, (i_k, b_k)$: this is an extension of marking (i, b) in which a string is marked based on the values of many bits.

destroy-marked : this removes all marked strings from the set S.

destroy-unmarked : this removes all unmarked strings from the set S.

unmark : this unmarks all marked strings in S.

test-if-empty : this operation determines whether the set S is empty or not. It is only executed at the end of a computation.

The main difference between this model and that of Adleman is in the manipulation of those DNA strands that are first immobilized on the surface. This approach greatly reduces the loss of DNA molecules during the purification steps [15] [61] [21] [26]. It also has the advantage of easy initial solution generation. The solution space (such as $0, 1^n$) can be synthesized both quickly and cheaply.

The major limitation of this model is that the scale of computation is severely restricted by the 2-dimensional nature of the surface based computation. To increase the scale, one must either a)increase the surface density, b)increase the surface area, or c) build linkage chemistry to extend out into solution from which the oligonucleotides can be attached, in order to make a local three-dimensional network on the surface.

B. Summary

This chapter has given a detailed explanation of why it is necessary to introduce a new DNA computing model. This new model should be fast, robust and error resistant. An analysis of the existing DNA computing models that are presented in this chapter has shown why these models cannot satisfy those requirements necessary to attract potential users.

CHAPTER III

THE ADVANCED DNA COMPUTING MODEL AND ALGORITHMS

The previous chapter described the reason existing DNA computing models cannot satisfy the requirements necessary for DNA computing's development. It has been consistently clear why it is necessary to have a new DNA computing model, and what the properties of that model should be. In this chapter, a new DNA computing model will be introduced on which new algorithms are developed. These new algorithms are presented as vehicles for demonstrating the advantages of the new model, and they can be expanded to solve many NP-complete problems. Those new algorithms can significantly speed up computation and therefore achieve a better time performance for DNA computing. With the given resources, these algorithms can solve problems of a much larger size than those possible with existing DNA computing algorithms. Error rates can be greatly reduced by applying these new algorithms [62]. Furthermore, the new algorithms have the advantage of dynamic updating, so an answer can be changed based on modifications to the initial condition [63]. In addition, these algorithms have the advantage of decoding all of the strands in the final pool both quickly and efficiently [64]. All the advantages provided by the new model make DNA computing very efficient and attractive in solving computationally intense problems.

A. Our New Model

Our new model adopts only mature DNA biological operations [5]. The following basic principle operations: *synthesis*, *ligation*, *separation*, *combination* and *detection* are selected for building the new model.

synthesis $I(P, \pi)$ To generate a pool of coded strands, P, following criteria π. Strands

are coded differently for different applications by using the four base nucleotides: A, G, T and C. A set is defined as a group of strands, and the container holding a set of strands is called a pool. If the criteria are the colors of a node in a graph, then a pool of strands coding all of those possible colors for that node is expected after *synthesis*. In the graph coloring problem, the strand is encoded for the colors of a set number of nodes. Here, a few consecutive nucleotides on the strand coded for the color of one node form a region. For example, in Figure 4, one strand consists of three regions such that $s = \{RBR\}$ where $(CCAAG)$, $(AATTC)$ and $(CCAAG)$ represent the colors for those three corresponding nodes as $R(Red), B(Blue)$ and $R(Red)$, respectively.

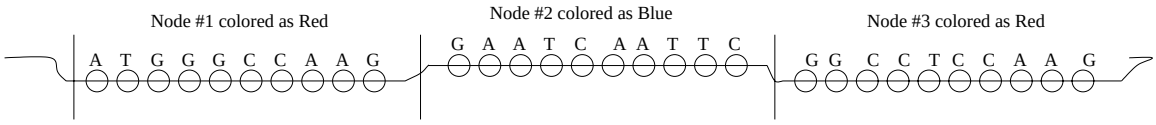


Fig. 4.: AN EXAMPLE OF THREE NODES IN A GRAPH THAT ARE COLORED BY R(RED), B(BLUE) AND R(RED)

ligation $L(P_3, P_1, P_2)$ To bind strands in pool P_1 with strands in pool P_2. Each code s_{1_i} in P_1 is *ligated* to every other code s_{2_j} in P_2. If the strands in P_1 represent the codes $\{s_{1_i} | i = 1, 2, \cdots, c, where\ s_{1_i} \in P_1\}$ and those in P_2 represent the codes $\{s_{2_j} | j = 1, 2, \cdots, d, where\ s_{2_j} \in P_2\}$, after the *ligation*, the *ligated* strands are stored in P_3. They represent the codes $\{s_k | k = 1, 2, \cdots, c \times d\}$, where $s_k = s_{1_i} s_{2_j}$ for $k = i + (j-1) \times c$.

separation $S(P, P_t, P_f, \theta)$ *Separation* is used to partition strands in pool P, and to

store those strands in two new pools: P_t and P_f based on criteria θ. After each *separation* operation, the strands that meet the criteria will be stored in one pool, P_t, while all strands that do not meet the criteria will be stored in the other pool, P_f. In order to perform the *separation* operation, many identical short strands defined as probes are attached to magnetic beads. These probes are then put into the pool containing the strands to be *separated*. Each probe can be paired up with a complementary strand in order to form a double helix. Such pair-ups only occur under the WC(Watson-Crick) complement rule: A only pairs with T and G only pairs with C. For example, in Figure 4, if the strands containing the region for node 1 which are colored 'R' need to be *separated*, the DNA short strands $TACCCGGTTC$ should be used as a probe because $TACCCGGTTC$ complements $ATGGGCCAAG$. Also, the double helix can be separated by heating in order to make paired strands part from each other without breaking the chemical bonds that hold the nucleotides of a single strand together. The strands in the pool which contain a region that complements the probes will be hybridized to and captured by the probes, while all those without the region will remain in the pool [34].

A gel-based *separation* technique for DNA computing [33] has been developed which uses gel-layer probes instead of beads to capture strands. The capture layer only retains the strand with a region that complements the probe when it is cooled down, and lets all strands pass when the layer is heated. The advantage gel-based probes has over bead-based probes is that the gel-based method is more accurate for capturing DNA molecules. Figure 5 illustrates the gel-based *separation*; a set of strands runs from the left side buffer to the right. At each capture layer, the temperature is kept cold in order to capture

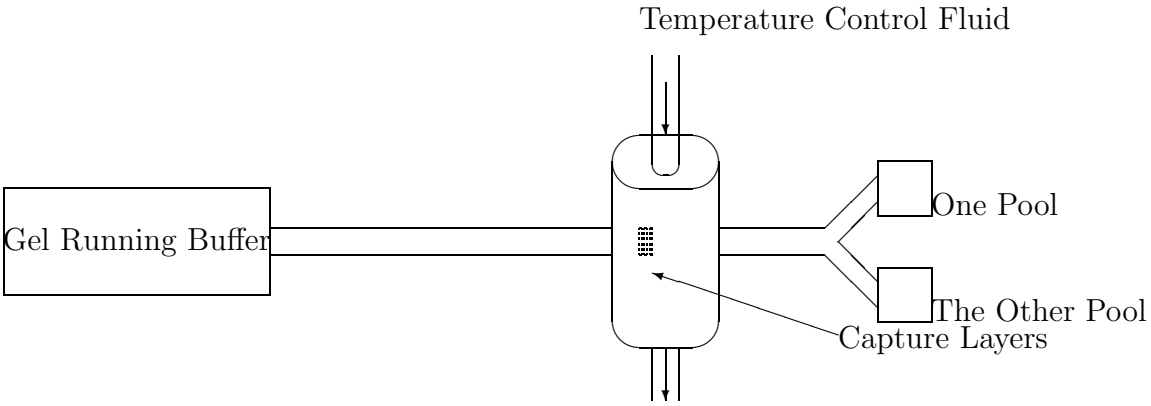


Fig. 5.: SEPARATION OPERATION BASED ON GEL LAYERS

the desired strands. All unwanted strands are then passed through into a single pool. The temperature is then raised to let all desired strands in the layer pass into another pool. The strands from the left buffer are *separated* and stored in two different pools.

combination $B(P, P_1, P_2)$ To pour two pools, P_1 and P_2, together to form a new pool, P.

detection $D(P)$ To check if there is any strand left in the pool, P. If the answer is "yes", the strands in the pool should be decoded.

B. The New Algorithms

The new algorithms for the 3-Coloring problem that were developed based on our new DNA computing model are used to demonstrate the advantages of this new model. The basic algorithm that significantly reduces computation time is introduced in this section. In the next section, the algorithm will be advanced to solve larger-sized problems, to dynamically update the answer, to lower the error rate, and to decode the final answers quickly and efficiently.

1. 3-Coloring Problem

The 3-Coloring problem, a special case of the k-Coloring problem where k=3, is a well known representative of the NP-complete problems class. A new algorithm for solving the 3-Coloring problem will be introduced here, and this introduction will simplify the following explanation of our new DNA computing model. The algorithms developed here can now be expanded to solve the k-Coloring problem and be generalized to solve other NP-complete problems.

k-Coloring Problem: A k-Coloring problem considers how to color an undirected graph $G = (V, E)$ in such a way that no two adjacent vertices share the same color [65]. Two nodes connected by an edge are referred to as adjacent vertices. The solution is the function $c : V \rightarrow 1, 2, \cdots, k$ such that $c(u) \neq c(v)$ for every edge $(u, v) \in E$. In other words, the numbers 1, 2, $\cdots$, k represent k colors, and the adjacent vertices must have different colors. The k-Coloring problem determines whether k colors are adequate to color a given graph [66].

A simple example graph with ten nodes and ten edges, G(10,10), is given in Figure 6. It is clearly shown there that the graph can be colored if $k \geq 3$.

Some existing DNA computing algorithms for solving the 3-Coloring problem can

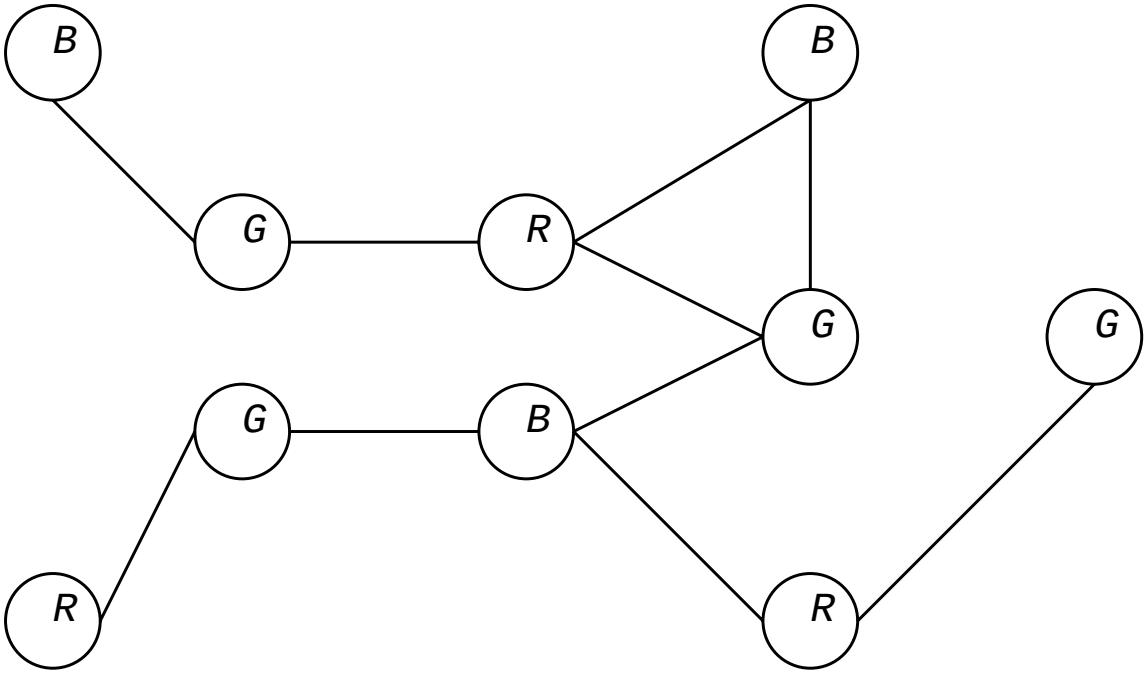


Fig. 6.: AN EXAMPLE GRAPH G{10,10} THAT CAN BE COLORED BY 3 COLORS: R(RED), G(GREEN) AND B(BLUE)

be found in [67]. Basically, all of these algorithms first generate a pool of encoded DNA strands that represent all possible color patterns for the n-node graph where each color pattern is an assignment of colors to nodes. For example, for nodes $n_1n_2n_3n_4$, "$BBRG$" is one pattern which assigns Blue to n_1, Blue to n_2, Red to n_3 and Green to n_4, while "$RGBB$" is another pattern which colors $n_1n_2n_3n_4$ as Red, Green, Blue and Blue, respectively. After the strands are generated and stored in a pool, those strands representing color patterns with no color conflict need to be *separated*. Any two nodes along an edge are defined as having a color conflict when they share the same color. In any color patterns with some color conflict that exist along some edges of the graph, the corresponding strands should be filtered out from the pool.

Our new algorithm is introduced next. Following that, different variations of the

algorithm and advantages to the new algorithm will be described.

2. The New Algorithm

Since here a given graph $G = (V, E)$, with $V = \{v_i | i = 1, 2, \cdots, n\}$ a set of nodes and $E = \{e_j | j = 1, 2, \cdots, m\}$ a set of edges, we took the divide and conquer approach to solving the 3-Coloring problem. We first partitioned graph G into two subgraphs: $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ such that $V_1 \cup V_2 = V$, $V_1 \cap V_2 = \phi$ and $\mid V_1 \mid \approx \mid V_2 \mid$ by eliminating all edges (u, v) such that $u \in V_1$ and $v \in V_2$. From this point forward, we will refer to this set of edges as the cut-set of G, C [65] [68]. The partition process was performed recursively. That is, subgraph G_i was partitioned into G_{2i+1} and G_{2i+2}, until each subgraph contained only one vertex and n subgraphs existed in total (See Figure 7).

When partitioning the graph G into n subgraphs, the algorithm will start to merge every two subgraphs both recursively and in parallel. Before they merge, every subgraph should be colored with 3 colors. During the merge, the color patterns of the two subgraphs can be combined together if no new color conflict is caused. Note that to merge two subgraphs, the edges in the cut-set eliminated previously in the partition of the two subgraphs will need to be added back and each addition of such an edge will introduce a color conflict if the nodes it links to are of the same color. Hence, the color patterns that work for the subgraphs may not necessarily work for the merged graph after they are combined, and the combined color patterns might be eliminated. The merging operation should continue until graph G is re-established and those color patterns legitimate for it are found.

Our new algorithm for solving the 3-Coloring problem on a sparse graph is presented in Figure 8. The first for loop is used to generate n pools of strands to represent all possible color patterns for n subgraphs while each subgraph initially

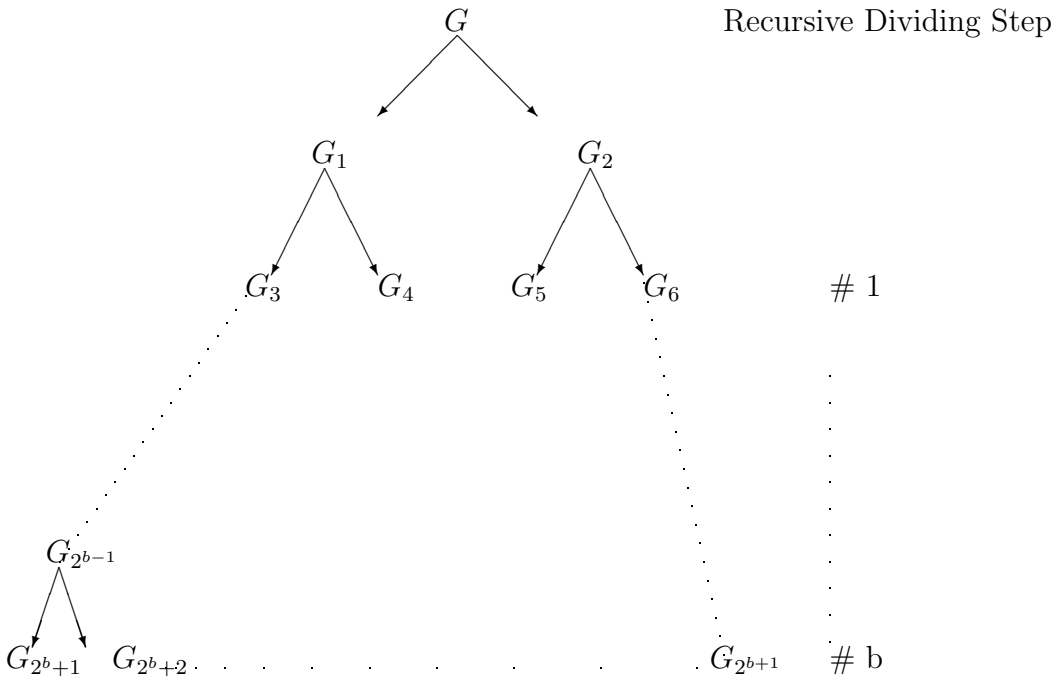


Fig. 7.: DIVIDE THE GRAPH, G_1, WITH $n = 2^b$ NODES UNTIL EACH SUB-GRAPH ONLY CONTAINS ONE NODE

Algorithm 1.

for i=*1 to* n **do**

  In Parallel($I(P_i$, *color of node* $i)$)

end

$f = n$

while $f \neq 1$ **do**

  In Parallel(*Make multiple copies of strands in all pools*)

  for *All odd* i **do**

    In Parallel($L(P_i, P_i, P_{i+1})$)

    In Parallel(*relabel all pools 1 to* $\frac{f}{2}$)

    for i =*1 to* $\frac{f}{2}$ **do**

      In Parallel(

      for $j = 1$ to E_i, E_i is the number of edges in C_i ***do***

        In Sequential { $S(P_i, P_{i_t}, P_{i_f}, \theta_{i_j})$ },

        θ_{i_j} *is the color conflicts along* e_j, $\forall e_j \in C_i$

      end

)

    end

  end

  $f = \frac{f}{2}$

end

Check if the pool is empty to conclude "yes" or "no" accordingly.

Fig. 8.: THE NEW DNA COMPUTING ALGORITHM FOR SOLVING THE 3-COLORING PROBLEM FOR SPARSE GRAPHS

only contains one node.

The function of the *while* loop is first to merge the pairs of two subgraphs. The bio-operation needed to merge the two subgraphs is *ligation*, which *ligates* strands in two pools to form longer strands. Let the color patterns for subgraph G_1 be s_i and those for G_2 be s_j. For any given s_i, all s_j's should be *ligated* with it, and such operations should be performed over all s_i's. That is, the strand for one color pattern of a subgraph is replicated and each duplicated copy is *ligated* with one of those strands representing the color patterns in the other subgraph. After they merge, all color patterns in the merged graph will be represented by *ligated* long strands.

After the merge, some *ligated* strands might encode color patterns with those color conflicts that were introduced by the edges in the cut-set eliminated in the partition step. Our task is to investigate every edge in the cut-set and detect all the color conflicts caused. This investigation is accomplished through the *separation* operation, i.e., in all the *ligated* strands, in order to filter out those strands that might contain any color conflicts from the pool. For any edge under investigation, two nodes, i and j, are connected. We must first *separate* the pool into three pools that contain the strands having node i colored R, G and B. In these three pools, the strands having node j colored R, G and B are respectively filtered out through the *separation* operation.

In the outer for loop, multiple copies of all strands in all pools need to be prepared for the following round of *ligation*. This duplication is accomplished through the PCR (Polymerase Chain Reaction) process [69].

If any strand is left in the final pool, then the 3-Coloring problem's answer is "yes". Otherwise, the graph cannot be colored only by three colors, and the answer is "no".

C. Advantages of the New Model and Extensions of the Fast Algorithm

1. Speeding Up The Process

A planar graph is a graph where no two edges cross one another and that is drawable on a plane. The size of the cut-set is $O(\sqrt{n})$ in such a graph. Our new algorithm can solve the 3-Coloring problem of the planar graphs within $O(log(n) + \sqrt{n})$ time. The first term, $O(log(n))$, is the time needed to merge the subgraphs recursively in order to form the original graph G. The second term, $O(\sqrt{n})$, is the time needed to *separate* the strands representing the legitimate color patterns of the graph from the pool. It has been shown already that in this type of case, our DNA computing algorithm has a shorter time performance than the existing algorithm. In what follows, we will present an advanced algorithm based on Algorithm 1 which speeds up the process and improves the time performance even further. Given a dense graph, the number of edges can be the number of vertices, squared. That means that the computation complexity of the existing DNA computing algorithms, $O(m + n)$, becomes $O(m) = O(n^2)$. Algorithm 2, the advanced DNA computing algorithm we propose, is shown in Figure 9. It is different from Algorithm 1 in that the color conflict is checked node by node, rather than edge by edge. All strands that represent color patterns with color conflicts between the node under investigation and all other nodes are isolated from the pool in a single step.

The implementation of this step can be accomplished by using the device shown in Figure 5, where probes in the capture layer represent the colors of all nodes connected to the node under investigation. All such nodes are checked simultaneously for color conflicts. The probes are different from what has been introduced previously, which then represented only the color of one node. For a graph with n nodes, $n - 1$ devices as given in Figure 5 are necessary for the $n - 1$ *separation* operations. As shown

in Algorithm 2, the time complexity of our new algorithm to solve the 3-Coloring problem on a dense graph of n nodes is $O(log(n) + n)$=$O(n)$, where the first term represents the complexity of combining the subgraphs necessary to regenerate the original graph, as well as the color patterns which are represented by the merged strands. The second term, $O(n)$, is the total complexity of checking all the n nodes for color conflicts, one node at a time. As compared to the $O(n^2)$ time complexity of the existing DNA computing algorithms, the time performance of our new algorithm offers a significant improvement.

A solution for the 3-Coloring problems of some graphs, may be more quickly reached when a pool becomes empty in the middle of the process. This means that if three colors are not sufficient to color even a subgraph of a graph, they are certainly not capable of coloring the entire graph, obviously leading then to the final answer of "no". The last step of the algorithm can be easily performed by the *detection* operation listed in the previous section.

Algorithm 2.

for i=*1 to* n **do**

| In Parallel($I(P_i$, *color of node* $i)$)

end

$f = n$

while $f \neq 1$ **do**

| In Parallel(*Make multiple copies of strands in all pools*)

| **for** *All odd* i **do**

| | In Parallel($L(P_i, P_i, P_{i+1})$)

| | In Parallel(*relabel all pools 1 to* $\frac{f}{2}$)

| | **for** i =*1 to* $\frac{f}{2}$ **do**

| | | In Parallel(***for*** $j = 1$ to N_i, N_i= # of nodes in subgraph i ***do***

| | | | *In Parallel(* $S(P_i, P_{i_t}, P_{i_f}, \omega_{i_j})$*),* ω_{i_j} *is the color conflicts along all edges with endpoint* j $\forall n_j \in V_i$

| | | ***end***

| | |)

| | **end**

| **end**

| In Parallel(*Make multiple copies of strands in all pools*)

| $f = \frac{f}{2}$

end

Check if the pool is empty to conclude the "yes" or "no" accordingly

Fig. 9.: THE NEW DNA COMPUTING ALGORITHM FOR SOLVING THE 3-COLORING PROBLEM FOR DENSE GRAPHS

2. Solving Larger Problems

The existing DNA computing algorithms for the 3-Coloring problems introduced have a solution space of 3^n, the total number of color patterns for n nodes with 3 colors, and a solution would require $O((n+m))$ operations [67]. The size of the largest problem solvable by these existing algorithms is greatly restricted by the solution space. In the previously published results of [67], the largest graph that can be correctly solved for the 3-Coloring problem using the existing DNA computing technique has 46 vertices, because the total number of color patterns must be smaller than the number of strands used to represent them within a liter of water. That is

$$3^n < 10^{22}$$

$$n \leq \lfloor log_3(10^{22}) \rfloor = 46$$

With our newly developed algorithm, this restriction is greatly loosened and consequently much larger problems can be solved. The size of the largest problem that can now be solved by our new algorithm is analyzed next.

The graph that can be colored with any number of colors is a graph with n nodes and no edges. An example of such a graph with no edges is shown in Figure 10. All other graphs with n nodes can be generated by adding edges to this disconnected graph.

Let r be the proportion of strands retained in the pool after each separation, based on any color conflicts introduced by adding one edge. After two sub-graphs are merged together, the edges in the cut set need to be added back. Every time an edge is added, some color patterns of the new graph may need to be dropped due to any color conflicts induced by the newly added edges. In other words, color patterns that work for the sub-graphs without the edge may contain color patterns

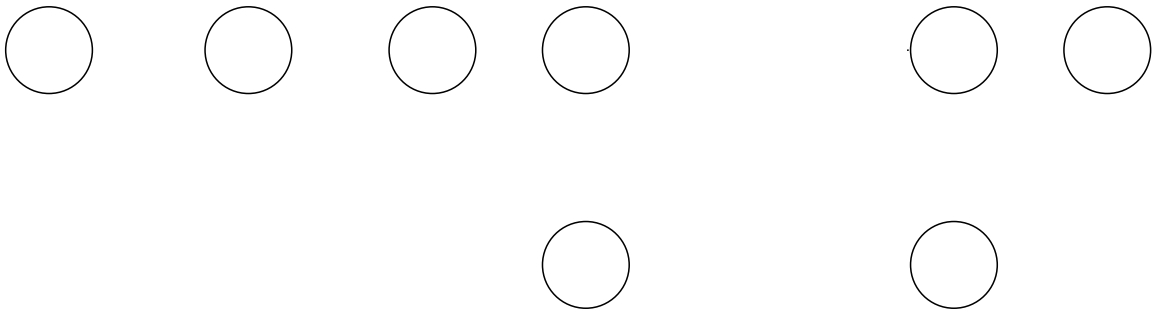


Fig. 10.: A DISCONNECTED GRAPH WITH NO EDGE

that have two nodes along an edge colored the same. Any color patterns that color two nodes with the same color will not work for the graph with the newly added edges. When the percentage of the strands being dropped from any of the sub-graph is 1-r, it results in 1-r out of the 3^n total combination of color patterns (for n nodes graph) being dropped with some of these color combinations never being generated. Based on a graph of n nodes with no edge, in order to reach a graph with m edges, m edges must be added. The edges in the cut-sets are added back when sub-graphs are merged to form larger graphs. The color patterns of the new graph are represented by longer strands. Out of the these strands (1-r^v) are dropped and r^v are retained after v edges are added in. This corresponds to (1-r^v) out of the 3^n total color patterns are dropped and r^v are retained. Among the strands that represent all possible color patterns of the graph, r^m can be kept in the final pool after m *separation* operations are performed for m newly added edges. This proportion of strands must be smaller than the total number of strands involved in the computation, e.g., 10^{22} in one liter of water. This is true for both the final pool and all the intermediate pools starting from n pools with strands in each representing the possible color patterns of each node. After the size of the strands grows when two sub-graphs are merged together, it decreases when edges are added in and some color patterns are dropped. In order

to keep the size of the strands representing the color patterns smaller than the total number of strand involved in the computation, the following restrictions apply. When $u \leq 45, 3u < 10^{22}$, there is no requirement for the number of edges; when $u > 45$, $3^u \times r^v < 10^{22}$, the number of edges must be large enough to make this equation stand. For the final pool it is:

$$10^{22} > 3^n * r^m.$$

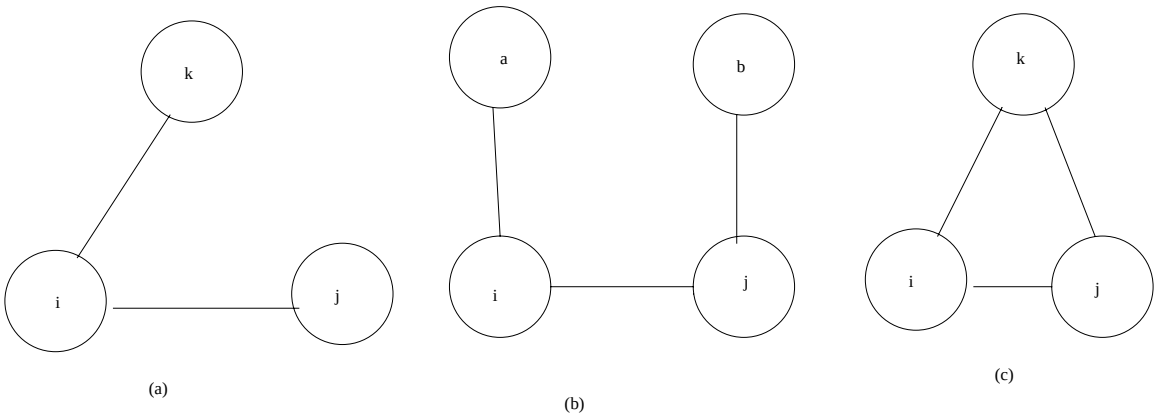


Fig. 11.: CONNECTIVITY OF i & j AND PROCESSED NODES

The implementation must check for color conflict one edge at a time. Without losing generality, we can assume that all adjacent edges sharing the same endpoint, node 1, are processed first. Those edges with endpoint 2 and so forth are processed one by one. Suppose that the edge connected to node i is under investigation and the other end is connected to node j, where $1 \leq i \leq n-1$ and $j > i$. To color one node, let the sample space be $S : \{s_1, s_2, \cdots, s_k\}$ for k colors. The probability $P(S) = 1$ and

that gives $P(s_i) = P(s_2) = \cdots = P(s_k) = \frac{1}{k}P(S) = \frac{1}{k}$. For the 3-Coloring problem in which $k = 3$ and $S : \{R, G, B\}$, we can say that, $P(R) = P(G) = P(B) = \frac{1}{3}$. To color multiple nodes, let c_j be the color of node j, and $P(c_j = s_i)$ be the probability that any node j is colored as s_i. The average proportion r can be calculated based on the following independent cases that cover all possibilities.

Case 1: Nodes i and j are connected to no other previously processed nodes. This means that no node k connects to either i or j, where $k < i$. The probability of the strands that then need to be kept is

$$\begin{aligned} P(c_i \neq c_j) &= P\{[(c_i = R)(c_j \neq R)] \bigcup [(c_i = G)(c_j \neq G)] \bigcup [(c_i = B)(c_j \neq B)]\} \\ &= \frac{1}{3} \times \frac{2}{3} + \frac{1}{3} \times \frac{2}{3} + \frac{1}{3} \times \frac{2}{3} = \frac{2}{3} \end{aligned} \tag{3.1}$$

Under this condition, color patterns $\{n_1 n_2 \cdots n_i \cdots n_j \cdots n_n\} = \{XX \cdots c_i \cdots c_j \cdots X\}$, where $c_i \in \{R, G, B\}$, $c_j \in \{R, G, B\}$ and $X \in \{R, G, B\}$, with $(c_i, c_j) \in \{(R, R), (G, G), (B, B)\}$ should be eliminated and those with $(c_i, c_j) \in \{(R, G), (R, B), (G, R), (G, B), (B, R), (B, G)\}$ should be kept. The proportion of those strands that need to be *separated* under this condition is $\frac{1}{3}$, and that to be kept is $\frac{2}{3}$.

Case 2: Either i or j is connected to at least one of the previously processed nodes k where $k < i$, as in the example in Figure 11(a). The color patterns $\{n_1 n_2 \cdots n_k \cdots n_i \cdots n_j \cdots n_n\} = \{XX \cdots c_k \cdots c_i \cdots c_j \cdots X\}$ with $c_i = c_j$ should then be eliminated. At this point, those strands that represent color patterns where $c_k = c_i$ have already been *separated* due to the connecting edge $e(k, i)$. Because color patterns where $c_i \neq c_j$, given that $c_i \neq c_k$, are kept and those with $c_k \neq c_i = c_j$ are *separated*, the probability that those strands will be kept is

$$P[(c_i \neq c_j) | (c_i \neq c_k)] = \frac{P[(c_i \neq c_j)(c_i \neq c_k)]}{P(c_i \neq c_k)} \tag{3.2}$$

Since $P(c_j \neq c_k) = 1 - P(c_j = c_k) = 1 - \frac{1}{3} = \frac{2}{3}$ and the event $(c_i \neq c_j)$ is independent from event $(c_i \neq c_k)$,

$$P[(c_i \neq c_j) \bigcap (c_i \neq c_k)] = P(c_i \neq c_j)P(c_i \neq c_k) = \frac{2}{3} \times \frac{2}{3},$$

and the value of (3.2) is

$$\frac{(\frac{2}{3} \times \frac{2}{3})}{\frac{2}{3}} = \frac{2}{3}$$

The strands coding $(c_k, c_i, c_j) \in \{(R,G,R),(R,G,B),(R,B,G),(R,B,R),(G,R,G),(G,R,B),(G,B,R),(G,B,G),(B,R,G),(B,R,B),(B,G,R),(B,G,B)$ are kept and those coding $(c_i, c_j, c_k) \in \{(R,R,G),(R,R,B),(G,G,B),(B,B,G)\}$ are *separated*, which indicates that the proportion r for this case is $\frac{2}{3}$.

Case 3: Nodes i and j are both connected to previously processed nodes of different colors (if the two processed nodes are of the same color, see case 4), and no node k connects to both of them, where $k < i$. Assuming that node i is connected to node a and node j is connected to node b, where $\{a, b\} < i < j$ and $c_a \neq c_b$, $a \neq b$, as in the example shown in Figure 11(b), the color patterns $\{n_1 n_2 \cdots n_a \cdots n_b \cdots n_i \cdots n_j \cdots n_n\}$ $= \{XX \cdots c_a \cdots c_b \cdots c_i \cdots c_j \cdots X\}$ with $c_i = c_j$ should be *separated* and those with $c_i \neq c_j$ should be kept. At this point, those color patterns containing only those nodes where $c_a \neq c_i$ and $c_b \neq c_j$ because those with $c_a = c_i$ or $c_b = c_j$ are eliminated due to presence of edges $e(a, i)$ and $e(b, j)$. The probability for those strands to be kept is

$$\begin{aligned} &P[(c_i \neq c_j)|(c_i \neq c_a)(c_j \neq c_b)] \\ &\quad = \frac{P[(c_i \neq c_j)(c_i \neq c_a)(c_j \neq c_b)]}{P[(c_i \neq c_a)(c_j \neq c_b)]} \end{aligned} \tag{3.3}$$

There are, in total, $\binom{3}{1} \times \binom{3}{1} \times \binom{3}{1} \times \binom{3}{1}$ possibilities of $\{c_i, c_j, c_a, c_b\}$ where each node can choose a color from $\{R, G, B\}$. In order to meet the criteria $(c_i \neq c_j)(c_i \neq c_a)(c_j \neq c_b)$,

the color for any node a will be picked first from $\binom{3}{1}$ different possibilities. Then $\binom{2}{1}$ different color options are left for node i, due to $c_a \neq c_i$. $\binom{2}{1}$ color selections are left for node j where $c_i \neq c_j$, and $\binom{2}{1}$ for node b to satisfy $c_j \neq c_b$. Therefore,

$$\begin{aligned} & P[(c_i \neq c_j)(c_i \neq c_a)(c_j \neq c_b)] \\ &= \frac{\binom{3}{1} \times \binom{2}{1} \times \binom{2}{1} \times \binom{2}{1}}{\binom{3}{1} \times \binom{3}{1} \times \binom{3}{1} \times \binom{3}{1}} \\ &= \frac{8}{27} \end{aligned}$$

where the numerator is the number of possibilities of $\{c_i, c_j, c_a, c_b\}$ that could meet the criteria $(c_i \neq c_j)(c_i \neq c_a)(c_j \neq c_b)$, and the denominator is the total possibilities of $\{c_i, c_j, c_a, c_b\}$.

$$\begin{aligned} & P[(c_i \neq c_a)(c_j \neq c_b)] \\ &= P[(c_i \neq c_a)]P[(c_j \neq c_b)] \\ &= \frac{2}{3} \times \frac{2}{3} = \frac{4}{9}. \end{aligned}$$

Thus, the value of (3.3) is

$$\frac{\frac{8}{27}}{\frac{4}{9}} = \frac{2}{3},$$

and the proportion of strands that should be kept is $\frac{2}{3}$.

Case 4: The last case is where there is at least one node k where $k < i$, and it connects to both nodes i and j, as shown in Figure 11(c). The color patterns $\{n_1 n_2 \cdots n_k \cdots n_i \cdots n_j \cdots n_n\} = \{XX \cdots c_k \cdots c_i \cdots c_j \cdots X\}$ where $c_i = c_j$ should be *separated*. At this point, these strands representing color patterns where $c_k = c_i$ and $c_k = c_j$ have already been *separated* due to the presence of edges $e(k, i)$ and $e(k, j)$. Because those color patterns $\{n_1 n_2 \cdots n_k \cdots n_i \cdots n_j \cdots n_n\} = \{XX \cdots c_k \cdots c_i \cdots c_j \cdots X\}$ where $(c_i, c_j, c_k) \in \{(R, G, B), (R, B, G), (G, R, B), (G, B, R), (B, R, G), (B, G, R)\}$ should be kept, and those with $(c_i, c_j, c_k) \in \{(R, R, G), (R, R, B), (G, G, R),$

$(G,G,B),(B,B,R),\ (B,B,G)\}$ should be *separated*, the proportion r is

$$\begin{aligned} & P\{(c_i \neq c_j)|[(c_i \neq c_k)(c_j \neq c_k)]\} \\ & \quad = \frac{P[(c_i \neq c_j)(c_i \neq c_k)(c_j \neq c_k)]}{P[(c_i \neq c_k)(c_j \neq c_k)]}. \end{aligned} \tag{3.4}$$

because

$$\begin{aligned} & P[(c_i \neq c_j)(c_i \neq c_k)(c_j \neq c_k)] + P[(c_i = c_j)(c_i \neq c_k)(c_j \neq c_k)] \\ & \quad = \quad P[(c_i \neq c_k)(c_j \neq c_k)] \\ & \quad = \quad P[(c_i \neq c_k)]P[(c_j \neq c_k)] \\ & \quad = \quad \frac{2}{3} \times \frac{2}{3}, \end{aligned}$$

and

$$P[(c_i \neq c_j)(c_i \neq c_k)(c_j \neq c_k)] = P[(c_i = c_j)(c_i \neq c_k)(c_j \neq c_k)],$$

The obvious conclusion is

$$P[(c_i \neq c_j)(c_i \neq c_k)(c_j \neq c_k)] = \frac{2}{9}.$$

Hence, the value of (3.4) is

$$\frac{\frac{2}{9}}{\frac{2}{3} \times \frac{2}{3}} = \frac{1}{2} \tag{3.5}$$

where nodes i and j are connected to one common node k, and $k < i$.

A tight boundary can be defined for keeping the strands in the pool at the time an edge is under consideration. After each edge is considered, at least one third of the strands should be *separated* from the current pool. At most, two thirds of the strands would then be retained in the pool to represent those color patterns in the graph that have no color conflicts.

For example, for a planar graph with 256 vertices, in order to generate an answer to the 3-Coloring problem with 10^{22} strands, the number of edges must be greater than 569. This is because

$$10^{22} > 3^{256} * r^m \Longrightarrow$$

$$m > 569$$

Because $\frac{569}{256} \approx 2$, 2 is the average minimum number of degrees required for each node in order to allow the 3-Coloring problem to be solved by one liter of strands. Meanwhile, 256 vertices are significantly greater than the number of vertices in a graph that is solvable by the existing 3-Coloring DNA computing algorithms, within the same given solution space. If an electronic computer that can perform 10^6 operations per second is used, then 10^{19} years is necessary to solve the 3-Coloring problem for a graph with 256 vertices, even if the fast Biegel and Eppstein algorithm [67] is used. By introducing our new model, it will take approximately 2 days to finish the implementation of the new algorithm, assuming that the average DNA operation takes 20 minutes [67]. Our new algorithm can be used to solve the 3-Coloring problems for graphs containing a significantly higher number of vertices and, as compared to the existing DNA computing algorithms or the algorithms designed for electronic computers, our algorithm is also significantly faster.

3. Error Resistance

At the time that DNA computing was introduced, a question was raised about how errors might affect the computing results. Although mature biological operations usually have a very low error rate, errors might still accumulate over time and thus might be responsible for incorrect answers. An introduction to our new algorithm with its error resistance based on new DNA computing model, follows.

Most of the errors in DNA computing occur during the *separation* operation. During the *separation* operation, one pool is *separated* into two pools. Let the pool containing all of those strands code color patterns possible for coloring the graph to be defined as the positive pool, P_t. The pool containing all of those strands that represent color patterns with color conflicts between a pair of nodes under investigation to be called the negative pool, P_f. These two pools may not be perfectly divided and, due to errors, might contain strands that should be included in the other pool. There are two kinds of errors: false positives and false negatives. False positive errors occur when strands containing color conflicts are selected to be placed in the positive pool. False negative errors occur when strands with no color conflicts are left in the negative pool. False positive errors are easy to handle because in the graph coloring problem, at the time the final pool is generated, strands will be decoded from the pool. Unwanted color patterns will be quickly dropped after they are checked using electronic computers for whether they work in the graph. On the other hand, false negative errors are more difficult to detect and they are usually more expensive to correct.

Presented below is our new DNA computing algorithm for solving the 3-Coloring problem that reduces the false negative error to a minute rate, ϵ. Assume that each *separation* operation has an average false negative rate of q. That is,

$$q = \frac{\alpha}{\alpha + \beta}$$

where α is the number of strands representing those color patterns with no color conflict that have been left in the negative pool, and β the number of strands representing those color patterns with no color conflict left in the positive pool. The proportion of strands representing color patterns with no color conflict kept successfully in the

positive pool is p, and

$$p = \frac{\beta}{\alpha + \beta}$$

where

$$p + q = 1.$$

The most straightforward method of reducing the false negative error rate is to repeat the same process a number of times. Suppose that the process is repeated d times and the false negative error rate that results is E, when $E = q^d$. To assure that $E \leq \epsilon$, we have $q^d \leq \epsilon$ and $d = \lceil log_q \epsilon \rceil$. That is, after repeating the *separation* operation $\lceil log_q \epsilon \rceil$ times, the false negative error rate should be smaller than ϵ. However, this method is not only inefficient but might also increase the false positive rate, thus, leading to strands with color conflicts being left in the positive pool.

Our previously described new algorithm can be advanced in order to reduce the false negative error rate. As we have previously discussed, the negative pool, P_f, might contain some strands that represent color patterns without color conflicts.

Instead of discarding these strands, the P_f pool should be further processed in the next operation, in order to retain those strands that might represent a final answer to the problem. After the second *separation* operation, P_f should be divided into the positive pool, P_{f_t}, and the negative pool, P_{f_f}. Pool P_t should then be *separated* into the positive pool, P_{t_t}, and the negative pool, P_{t_f}, where P_{t_t} contains all strands currently representing color patterns with no color conflict. P_{t_f} would then contain those strands representing color patterns that have color conflicts along the new edge under consideration. Pools P_{t_f} and P_{f_t} should then be combined and labeled as P_{f_1}. P_{t_t} is then relabeled as P_t, and P_{f_f} is labeled as P_{f_2}. After subsequent *separation* operations at different levels, the corresponding processes are listed in Figure 12. Pool P_{f_1} contains all those strands representing color patterns with color conflict along at least one edge, while pool P_{f_2} contains those strands with conflicts along at least two different edges. P_t contains strands that represent those color patterns capable of coloring the graph. The possibility that pool P_{f_1} has strands that should be in P_t is q where q is the false negative rate defined above. The possibility that pool P_{f_2} has strands that should be in P_t is q^2. The same operations should continue until $d+1$ different pools, which are $P_{f_1}, P_{f_2}, \ldots, P_{f_d}$ and P_t, are generated where $q^d \leq \epsilon$. The false negative rate for those strands left in pool P_{f_d} that represent color patterns with no color conflict is now smaller than ϵ. The extra expense required to achieve this lower error rate from our new algorithm is very small. With a 1% false negative error rate [70] in a single *separation* operation, it is very easy to reduce the overall false negative rate to 0.0001%, with d being as small as 3.

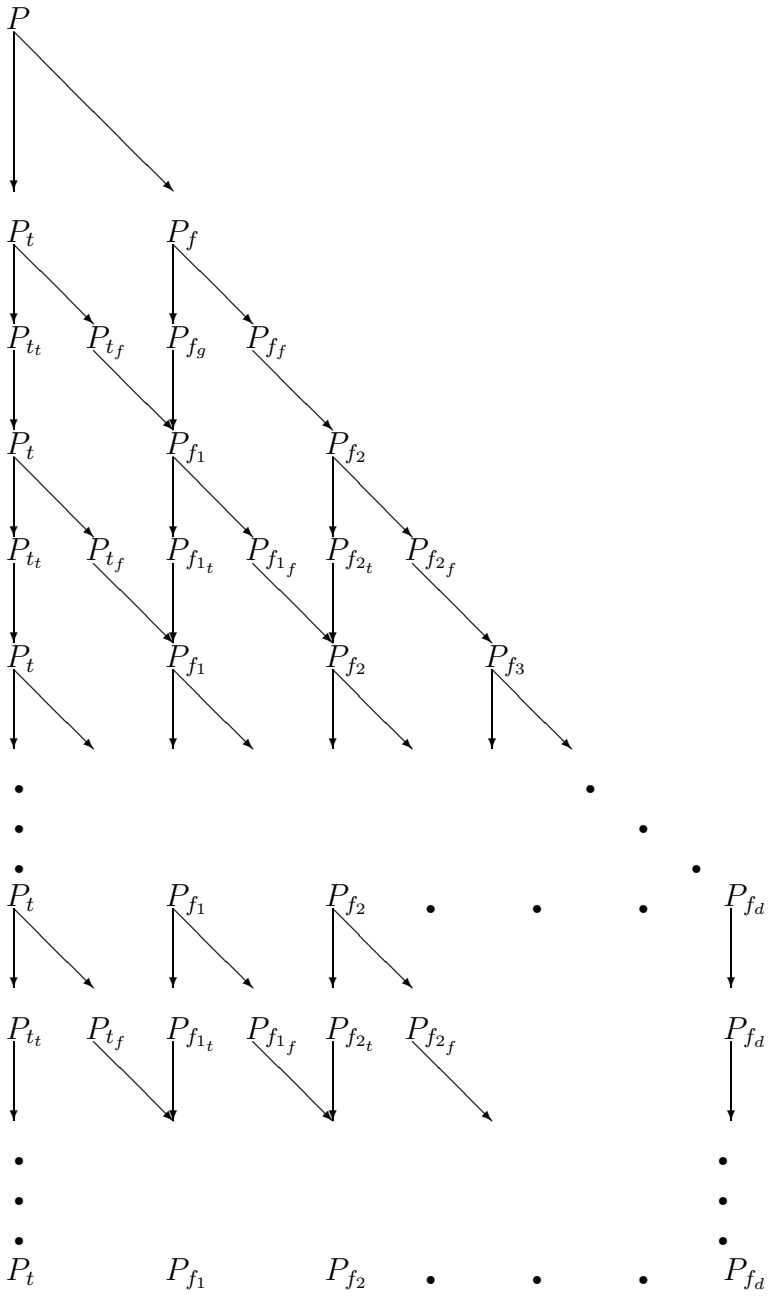


Fig. 12.: *SEPARATING* THE POOL P INTO d+1 DIFFERENT POOLS

4. Dynamically Updating Algorithms

Once a solution to the 3-Coloring problem of the graph is obtained, it is significant to have a method that can quickly update the solution without restarting the algorithm and completely recalculating if minor changes in the initial conditions are necessary. The following is an effort toward making such a dynamically updating solution both realistic and efficient by using the new DNA computing model.

In 3-Coloring problems, four possible changes may occur in the initial condition: both nodes or edges could be either inserted or removed. Different strategies need to be considered to update the answer based on the originally generated "yes" or "no" answer.

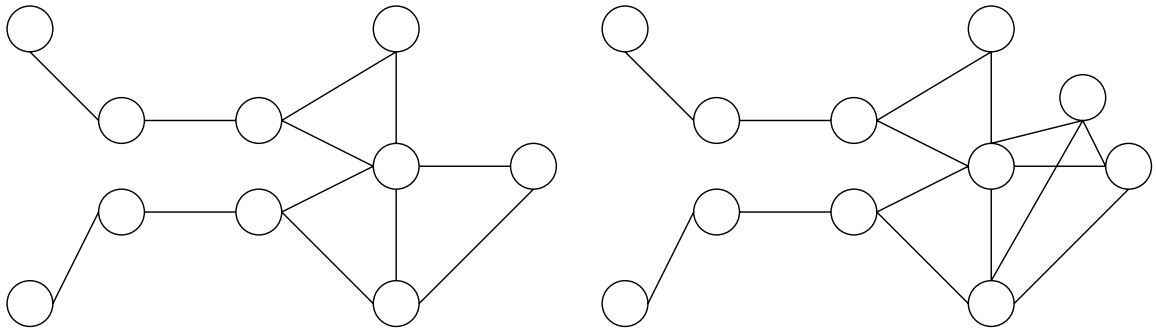


Fig. 13.: ADDING ONE NODE TO THE GRAPH

Let us begin with the easiest updating strategies. If the original answer is "yes" and an edge or node is removed from the original graph, the answer will remain "yes."

If the original answer is "no", it will remain "no" when nodes or edges are added in.

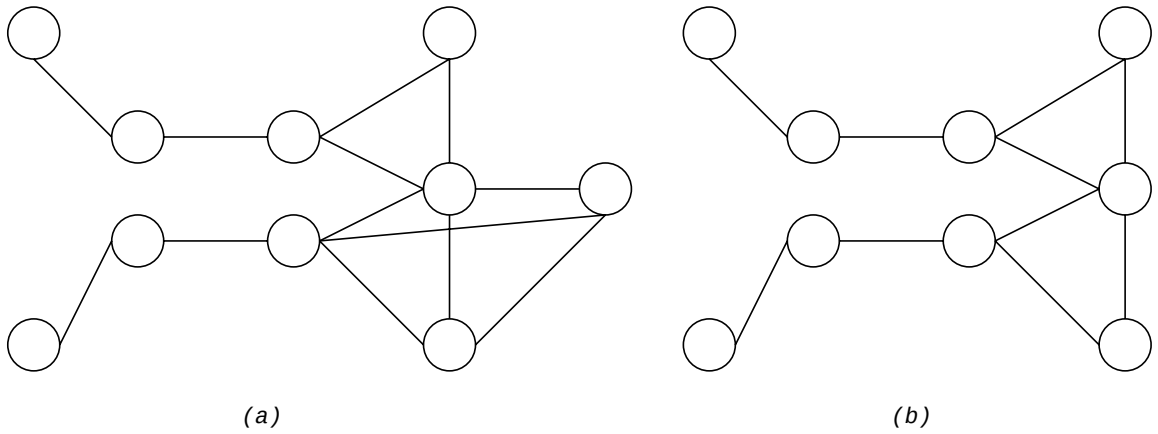


Fig. 14.: REMOVING ONE NODE FROM THE GRAPH

If the original answer is "yes," it may be changed to "no" after a node is inserted into the graph. An example is shown in Figure 13. In this figure, the answer to the 3-Coloring problem of the graph given in Figure 13(a) is "yes," but it changes to "no" once a node is inserted to form the graph shown in Figure 13(b).

If the original answer is "no," it may be changed to "yes" after a node is removed from the graph. Figure 14 illustrates an example of this. Figure 14(a) contains a graph with the answer "no" to the 3-Coloring problem. The answer changes to "yes" after one node is removed from the graph, as shown in Figure 14(b).

Inserting or removing an edge can be similarly dealt with because at least one edge must be eliminated if a node is removed, and at least one edge must be added if a node is inserted.

The following illustrates how to dynamically update a solution when a node or edge is inserted into the graph, following an original answer of "yes". The strands in the final set, P_t, are checked for possible new answers. The final set is the only set that can be used because it is the only set that contains strands that represent all possible coloring solutions that do not have any color conflicts among all nodes,

except the newly added one. Based on these sets, only those color conflicts that occur between the newly added node and the nodes connected with it need to be checked. In other words, only the newly added edges need to be checked for color conflicts.

The most difficult case occurs when a node or edge is removed from a graph with an original answer of "no". The answer to the new graph could be either "yes" or "no". Removing a node includes removing both the node itself and all edges connecting that node to the graph. The dynamically updating algorithm for this case is as follows: the DNA computing result that reflects an original answer of "no" is represented by an empty P_t set with no strand. All other sets then represent the coloring patterns of the original graph with the color conflicts. After removing nodes or edges, some coloring patterns may no longer have conflicts. The task, then, is to identify those patterns represented by the DNA strands. Note that the strand sets to be examined are limited in number. Only those strands representing color patterns with color conflicts involving the pair of nodes connected by the removed edges need to be checked. Finding that particular strand sets takes $O(\alpha)$ steps, where α is the number of edges being removed. If α is not large this process is far less expensive than re-computing the updated graph from the very beginning.

The detailed algorithm needed to find the answer to the new graph with the removed edges, based on the original "no" answer, is illustrated in Figure 15.

When only one edge is removed from the original graph, pool P_{f_1} should be checked. This is because P_{f_1} contains all of those strands that represent the color combinations of the graph that have no color conflicts along all edges except one. Assuming that the two nodes along the removed edge are n_1 and n_2, the strands that need to be *separated* from the pool are those that have the two nodes colored as {RR}, {BB} and {GG}. This means that only those strands with two identically colored nodes must be extracted to a new pool, P_{new}. If P_{new} is not empty, the answer

Algorithm 3.

for *i=1 to* α **do**

| In Parallel($S(P_{f_i}, P_{new_i}, P_{f_i}, \theta_i)$, θ_i *is the color conflicts along* i *exact # of edges*)

end

$B(P_{new}, \phi, \phi)$

for *i=1 to* α **do**

| In Parallel($B(P_{new}, P_{new}, P_{new_i})$)

end

$B(P_{new}, P_t, P_{new})$

for *j=1 to* β **do**

| $S(P_{new}, P_{new}, P_{new_f}, \omega_i)$, ω_j is the color conflicts based on edge e_j

end

Check if pool P_{new} is empty to return a "yes" or "no" answer accordingly

Fig. 15.: THE DYNAMICALLY UPDATING ALGORITHM FOR THE 3-COLORING PROBLEM WHEN α EDGES ARE REMOVED AND β EDGES ARE ADDED

to the 3-Coloring problem for the new graph will be "yes", which is different from the original graph. Otherwise, the "no" answer remains.

When two edges are removed from the graph, both P_{f_1} and P_{f_2} need to be checked. This is because P_{f_2} may contain strands that represent color combinations with color conflicts along both removed edges. P_{f_1} may contain strands that represent those color combinations of the graph with a color conflict along only one of the two removed edges. Suppose the two removed edges are e_1 and e_2. Then those strands that need to be extracted from pool P_{f_2} using the *separation* operation must represent the color combinations of the graph that have color conflicts along **both** edges. The

strands that should be extracted from P_{f_1} are those that represent color combinations with color conflicts along **either** e_1 **or** e_2. The extracted strands are then stored in a new pool, P_{new}. If P_{new} is not empty, the answer to the 3-Coloring problem for the new graph will be "yes", which is different from the original graph. Otherwise, the answer for the 3-Coloring problem to the new graph remains "no".

When α different edges are removed from the original graph, α different pools should be checked. These pools are P_{f_1}, P_{f_2}, ..., P_{f_α}. For different pools, different operations need to be conducted. For pool P_{f_1}, all strands should be left, due to the color conflict along one edge. If the edge that causes the conflict is removed, the answer will change to "yes". Because of this, all strands in this pool representing those color combinations with color conflicts along one of the removed α edges should represent answers to the 3-Coloring problem of the new graph. For pool P_{f_2}, all strands representing color combinations that have color conflicts along two, and only two of the removed edges represent answers to the 3-Coloring problem of the new graph. For pool P_{f_t} where $t \leq \alpha$, all strands representing color combinations with color conflicts along exactly t different removed edges will generate answers to the 3-Coloring problem of the new graph. All strands extracted from these sets will be stored in a new pool, P_{new}. If P_{new} is not empty, the answer of the 3-Coloring problem for the new graph is "yes", and thus different from the original graph. The answer is "no" if P_{new} is empty.

When the graph is changed by both removing and adding edges, multiple processing steps need to be considered. Assuming that the number of edges being removed is α and the number of edges added is β, those strands with color conflicts along the removed edges should be found first. This will put the strands that are to be considered for the following operations into one pool, P_{new}, instead of involving several pools. Those α edges should first be examined by using the method introduced

above to go through α different pools. Then, P_t is *combined* with P_{new} and relabeled P_{new}. This is due to the fact that those strands that may generate the "yes" answer are distributed in $\alpha + 1$ different pools. Collecting the strands in one pool will save time and further operations, as compared to working on each pool, one at a time. If no strands are left in pool P_{new}, then the answer to the new graph is "no". If there are strands in pool P_{new} after α edges have been removed, the color conflicts along β edges must be checked. This operation can be accomplished in a manner similar to what has been described above for adding edges.

Compared to the existing algorithms, our new method can dynamically update the solution if the initial condition changes for the 3-Coloring problem of a graph. It can also solve the 3-Coloring problem for many similar graphs. The complexity of the existing algorithms is $O((m + n))$, where n is the number of vertices and m is the number of edges [67]. If our updating process is not used, any change in the initial condition must result in a restarting of the process. With our new algorithm, the number of extra processes that need to be conducted depends upon the the significance of the changes. The complexity of our updating process is $O(\alpha + \beta)$, where α is the number of edges being removed, and β the number of edges being added.

When this method is used to solve the 3-Coloring problem for multiple similar graphs, the time complexity is $O(\xi)$ after the solution for one graph is generated, where ξ is the difference between the number of edges of the two graphs.

In order to implement this process, it is necessary to check the extra space and effort necessary for making dynamic updating available. First, m additional containers are needed to keep m extra sets of strands. Second, the extra DNA material necessary for generating these sets needs to be contained. Because strands are generated to represent all color combinations for the graph before the *separation* process

takes place, no extra material is necessary (as compared to the existing algorithms) until the answer is generated for the original graph. Extra material is only necessary if new solutions need to be formed for the modified graph if edges and/or nodes are added.

When the procedure for approaching a 3-Coloring problem of a given graph is finished and a new graph is provided, one must then determine whether to start again from the beginning or to use the dynamic updating method to generate the new answer?

In order to make this decision, one must assume that the implementation of the algorithms introduced above for the 3-Coloring problem of the graph with n nodes and m edges has been finished, and the 3-Coloring problem of a new graph needs to be solved. This new graph must have N nodes and M edges. This graph can be converted from the existing graph by first removing δ nodes and α edges, and then adding γ nodes and β edges. The new graph can be generated by changing the original graph, or it can be treated as a totally new graph. In order to solve the problem for the new graph, N *ligation* and M *separation* operations are necessary if the algorithm is being restarted from the beginning. The total time necessary would then be:

$$T_1 = N \times l + M \times s$$

where l is the time for each *ligation* operation and s is the time necessary for each *separation* operation. Here, *combination* operations are ignored due to their simplicity because the time needed for these operations is very short, especially when compared to the other operations used in DNA computing. When the answer is produced based on the pools already generated using this new, dynamically updating

strategy, the time necessary for reaching the answer is:

$$T_2 = (\alpha + \beta) \times s + \gamma \times l$$

In order to take advantage of the new method, the time needed must be shorter than the time it would take to restart the algorithm from the beginning.

$$T_2 \leq T_1$$

$$(\alpha + \beta) \times s + \gamma \times l \leq N \times l + M \times s$$

$$(\alpha + \beta) \times s + \gamma \times l \leq (n + \gamma - \delta) \times l + (m + \beta - \alpha) \times s$$

because $N = n + \gamma - \delta$ and $M = m + \beta - \alpha$. It is easy to get

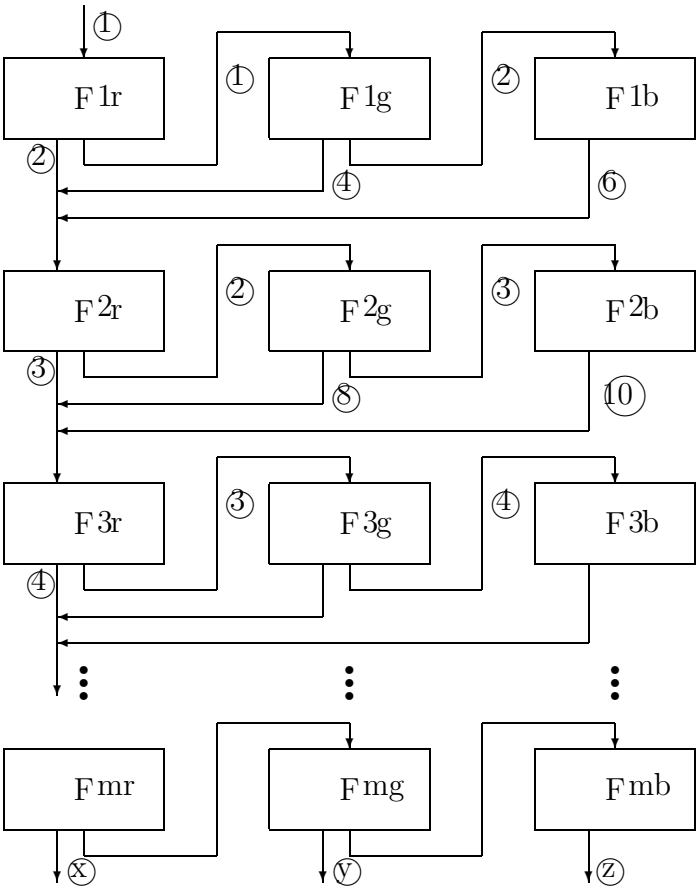
$$(m - \alpha) \times s + (n - \delta) \times l \geq \alpha \times s,$$

as $n - \delta$ is always greater than 0. The above condition can be tightly restrained as follows:

$$(m - \alpha) \times s > \alpha \times s$$

so that $\alpha < \frac{m}{2}$. The algorithm needs to be restarted from the beginning only when the change is significant – in other words, when more than half the edges need to be removed in order to generate a new graph from the original.

Given the above conclusion, it becomes evident that there is no need to retain all m sets. At least half of the pools can be destroyed in order to save storage space. This saves the expense once required for storing m sets of strands, and the material needed to work on them.



Each circled number represents the place where a valve is placed

Fig. 16.: AUTOMATED "*DECODING*" PROCESS WITH 3M FILTER

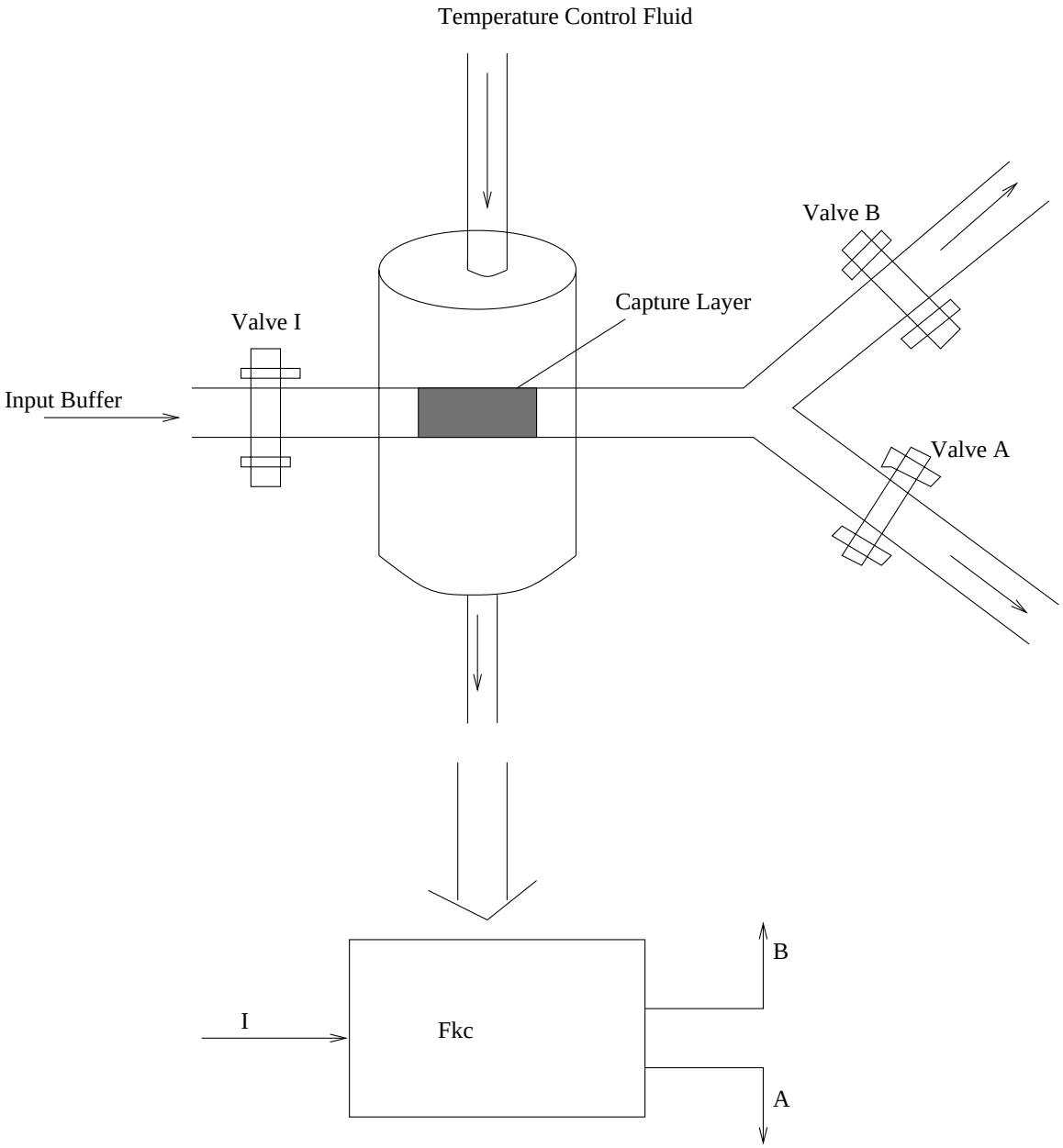


Fig. 17.: AN EXAMPLE OF A FILTER FOR THE *KTH NODE WITH COLOR C*

5. The Efficient Decoding Algorithm

After the final set containing all solutions to the 3-Coloring problem for the graph is generated, it is time to decode the strands in order to discover those color patterns that can correctly color the graph. Each strand in the pool has one answer encoded in it, and some strands in the pool may encode the same answer.

The new method introduced here can decode all color patterns represented by the DNA strands in pool P_g without using the electron microscope to "read" the strands one by one. It is much more cost and time efficient when compared to the method of decoding the strands one at a time by using an electron microscope. The flow diagram for this new method is illustrated in Figure 16. The function of each box in this figure is to represent a filter based on the gel-based *separation*. The detailed structure of each box is shown in Figure 17, and the filter function is given below: before the input buffer can be filled, the capture layer must be filled with small segments of DNA strands. Each filter is named Fkc, where $k \in \{1, 2, \cdots, m\}$ and $c \in \{R, G, B\}$. The capture layers contains those DNA strand segments that represent the complement of color c for node k. The temperature must first be reduced. The input buffer can then let the input pool flow into the capture layer, and valve A be opened. All strands that contain the segment representing color c for node k should be captured in the layer. The rest of the strands in the input pool will pass the layer and go through valve A. When this process is finished, valve B must be opened and the temperature of the container increased. All strands containing the segment that represents node k being colored with color c is *separated* from the rest of the pool. The order of the operations are indicated in Figure 16. At the time t_i, all valves labeled i must be opened and the temperature of the corresponding container either cooled down or warmed up. Eventually, x, y and z will provide some output sets. At

time t_{m+1}, x should output a set. This set should only have strands representing color combination of $N_nN_{n-1}\cdots N_1 = RR\cdots R$ for n nodes. For the following time t_{m+2}, t_{m+3}, $\cdots$, t_{m+3^m-2}, other sets containing strands representing the color combinations $N_nN_{n-1}\cdots N_1 = \{RR\cdots RG, RR\cdots RB, RR\cdots GR, RR\cdots GG, \cdots, RB\cdots BB\}$ will be outputted from x. The color combinations represented by strands will be outputted from y and z in the following order:
$N_nN_{n-1}\cdots N_1 = \{GR\cdots RR, GR\cdots RG, GR\cdots RB, GR\cdots GR, GR\cdots GG, \cdots, GB \cdots BB\}$ and $N_nN_{n-1}\cdots N_1 = \{BR\cdots RR, BR\cdots RG, BR\cdots RB, BR\cdots GR, BR\cdots GG, \cdots, BB\cdots BB\}$.

The *decoding* process has been simplified by using the *separate* and *detect* operations. At the time a set is outputted from x, y or z, the *detect* operation will check to see if it is empty. If not, the corresponding color combination is good for coloring the graph, and will have no color conflicts along any edge. Otherwise, if the set is empty, the corresponding color combinations cannot be used to color the graph.

The extra space and effort necessary to efficiently decode the strands in the final set are also greatly reduced. At the beginning, it seems that $3m$ different filters are needed. When the algorithm for generating the final set is implemented, it demonstrates that all of the necessary filters have already been generated in order to *separate* the initial pool containing those strands representing all color combinations. The extra effort is needed only to reorder these filters. After these filters are connected together, the valves and temperature of the containers can be automatically controled by an electronic microcontroller. This automation greatly reduces the involvement of human beings and it makes the DNA computing more error resistant. In addition, all of the filters in the far right column (as shwon in Figure 16) are not necessary because those strands coming into these strands will pass through the filter together. Hence, there is no filter function needed here. Storage buffers can be used to replace

these filters for temporarily storage in order to simplify the system. The other area in which effort is needed is the *detect* operation. This step can be accomplished very effectively and quickly.

D. Summary

This chapter has presented a new DNA computing model from which many DNA computing algorithms have been developed. The properties of this new model on which the new algorithms' development have been based include fast implementation time, the ability to solve larger problems, error resistance, dynamic updating and fast and efficient decoding capabilities. All of these advantages should make DNA computing more efficient and will attract new users seeking to solve computationally intense problems.

CHAPTER IV

CONCLUSION

This dissertation has presented a new model for DNA computing. Based on this new model, new algorithms for the 3-Coloring problems have been presented. This new algorithms represent a significant speed improvement over all existing algorithms.

Our algorithms were obtained by parallelizing the *separation* operation on multiple edges, and also by parallelizing other DNA computing processes. This provides the opportunity to solve very large problems which currently cannot be solved by electronic computers in any reasonable amount of time. The solution space of 10^{22} strands in a one liter pool can now efficiently be used. With the given solution space, problems of a large size that currently cannot be solved using existing DNA computing techniques are now solvable. The introduction of these new algorithms makes DNA computing a more attractive option to potential users who want to solve computational intense problems that are currently considered unsolvable.

Our new algorithm for error resistance has also been presented. DNA computing techniques have here been greatly improved by reducing the error rate to a considerably small percentage. This improvement will make DNA computing significantly more reliable.

In addition, these new algorithms have the advantage over existing algorithms of utilizing dynamic updating. These new algorithms represent a huge improvement over the existing algorithms. Instead of re-starting the DNA computing algorithm from the beginning every time the initial condition changes, this new method generates a new solution through only a few extra DNA operations, based on the existing answer. It can also quickly solve problems similar to those that have already been solved.

No extra material is needed to prepare for the dynamic updating process. The

only expense necessary is for extra containers in which the additional pools of DNA strands can be stored. As compared to the existing DNA computing algorithms, this new method can achieve a solution much more quickly after the answer for the first problem is generated. Finally, it is financially much more efficient.

These new algorithms decode all answers to the problem represented by DNA strands. This is a significant advantage over those methods that can locate only a few answers within the whole set. The *decoding* process of the newly introduced algorithm is very fast and efficient when compared to the existing method which uses electron microscopy to decode the strands. Instead of only providing a "yes" or "no" answer to the problem, the new model can provide exact answers. Regrading the *separation* operation, this new method can decode all strands in a set with very little extra cost. Therefore, these new algorithms represent a significant improvement over the naive search employed by existing algorithms.

Based on this new model, other algorithms can also be developed to solve different NP-complete problems, as well as those problems that are computationally intense. All of these algorithms that can be developed based on our new model will have the same advantages as described above. This new model, then, is able to expedite the development of important new DNA computing techniques. Consequently, this will make DNA computing more attractive to potential users who want to solve problems currently considered unsolvable.

A. Future Work

Special bio-operations need to be designed and/or identified in the near future in order to implement a divide-and-conquer approach in DNA computing (which will significantly reduce the time required by many DNA algorithms and improve their performance). Developing more efficient and accurate laboratory techniques will not

only benefit researchers in DNA computation, but will also yield positive benifits for molecular biologists. Second, simplifying the DNA computing algorithm design process so that both computer engineers and biological scientists can design efficient DNA computing algorithms with little training will also greatly improve this field. Third, automating the DNA computing algorithm implementation so that the DNA computing models and techniques can be integrated into a general purpose computer will surely yield significant advances in DNA computing, thus producing faster and more financially efficient solutions to computationally intense problems.

REFERENCES

[1] N. Forbes, "Biological inspired computing," *Computing in Science & Engineering*, vol. 2, pp. 83–87, Washington, Nov. - Dec. 2000.

[2] E. Brynjolfsson and L. Hitt, "Beyond computation: Information technology, organization transformation, and business performance," *Journal of Economic Perspectives*, vol. 14, no. 3, pp. 23–48, St. Paul, 2000.

[3] C. Mann, "The end of moore's law," *Technology Review*, vol. 103, no. 3, pp. 48, Cambridge, May/June 2000.

[4] J.H.M. Dassen, "DNA computing, promises, problems and perspective," *IEEE Potentials*, vol. 16, no. 5, pp. 27–28, Washington, Dec. 1997 - Jan. 1998.

[5] L. Adleman, "Molecular computation of solutions to combinatorial problems," *Science*, pp. 1021–1024, Washington, November 1994.

[6] J. Amenyo, "Mesoscopic computer engineering: Automating DNA-based molecular computing via traditional practices of parallel computer architecture design," in *Second Annual Meeting on DNA Based Computers*, Princeton, June 1996, pp. 217–235.

[7] Y. Gao, M. Garzon, R. C. Murphy, J. A. Rose, R. Deaton, D. R. Franceschetti, and S. E. Stevens Jr., "DNA implementation of nondeterminism," in *3rd DIMACS Workshop on DNA Based Computers*, Princeton, June 1997, pp. 204–211.

[8] G. Gloor, L. Kari, M. Gaasenbeek, and S. Yu, "Towards a DNA solution to the shortest common superstring problem," in *Fourth International Meeting on DNA Based Computers*, Philadelphia, June 1998, pp. 111–116.

[9] V. Gupta, S. Parthasarathy, and M J. Zaki, "Arithmetic and logic operation with DNA," in *3rd DIMACS Workshop on DNA Based Computers*, Philadelphia, June 1997, pp. 212–222.

[10] T H. Leete, M D. Schwartz, R M. Williams, D H. Wood, J S. Salem, and H. Rubin, "Massively parallel DNA computation: Expansion of symbolic determinants," in *Second Annual Meeting on DNA Based Computers*, Princeton, June 1996, pp. 49–66.

[11] N. Morimoto and M. Suyama, "Solid phase DNA solution to the Hamiltonian path problem," in *3rd DIMACS Workshop on DNA Based Computers*, Philadelphia, June 1997, pp. 83–92.

[12] J S. Oliver, "Computation with DNA-matrix multiplication," in *Second Annual Meeting on DNA Based Computers*, Princeton, June 1996, pp. 236–248.

[13] Z. Qiu and M. Lu, "Arithmetic and logic operations for DNA computers," in *Parallel and Distributed Computing and Networks (PDCN'98)*, Calgary, AB, Canada, December 1998, pp. 481–486.

[14] M. Amos, A. Gibbons, and D. Hodgson, "Error-resistant implementation of DNA computations," in *Second Annual Meeting on DNA Based Computers*, Princeton, June 1996, pp. 87–101.

[15] W. Cai, A E. Condon, R M. Corn, E. Glaser, Z. Fei, T. Frutos, Z. Guo, M G. Lagally, Q. Liu, L M. Smith, and A. Thiel, "The power of surface-based DNA computation," in *RECOMB'97. Proceedings of the First Annual International Conference on Computational Molecular Biology*, New York, 1997, pp. 67–74.

[16] J. Chen and D. Wood, "A new DNA separation technique with low error rate,"

in *3rd DIMACS Workshop on DNA Based Computers*, Philadelphia, June 1997, pp. 43–58.

[17] M. Deputat, G. Hajduczok, and E. Schmitt, "On error-correcting structures derived from DNA," in *3rd DIMACS Workshop on DNA Based Computers*, Philadelphia, June 1997, pp. 223–229.

[18] D. Wood, "Applying error correcting codes to DNA computing," in *Fourth International Meeting on DNA Based Computers*, Philadelphia, June 1998, pp. 109–110.

[19] T. Yoshinobu, Y. Aoi, K. Tanizawa, and H. Iwasaki, "Ligation errors in DNA computing," in *Fourth International Meeting on DNA Based Computers*, Philadelphia, June 1998, pp. 245–246.

[20] D. Boneh, C. Dunworth, J. Sgall, and R J. Lipton, "Making DNA computers error resistant," in *Second Annual Meeting on DNA Based Computers*, Princeton, June 1996, pp. 102–110.

[21] Q. Liu, A. Frutos, L. Wang, A. Thiel, S. Gillmor, T. Strother, A. Condon, R. Corn, M. Lagally, and L. Smith, "Progress towards demonstration of a surface based DNA computation: A one word approach to solve a model satisfiability problem," in *Fourth International Meeting on DNA Based Computers*, Princeton, June 1998, pp. 15–26.

[22] Q. Liu, Z. Guo, A E. Condon, R M. Corn, M G. Lagally, and L M. Smith, "A surface-based approach to DNA computation," in *Second Annual Meeting on DNA Based Computers*, Princeton, June 1996, pp. 206–216.

[23] R. Deaton, R. C. Murphy, M. Garzon, D. R. Franceschetti, and Jr. S. E. Stevens,

"Good encodings for DNA-based solutions to combinatorial problems," in *Second Annual Meeting on DNA Based Computers*, Princeton, June 1996, pp. 131–140.

[24] A G. Frutos, A J. Thiel, A E. Condon, L M. Smith, and R M. Corn, "DNA computing at surfaces: 4 base mismatch word design," in *3rd DIMACS Workshop on DNA Based Computers*, Philadelphia, June 1997, pp. 238–239.

[25] M. Garzon, R. Deaton, P. Neathery, R.C. Murphy, D. R. Franceschetti, and S.E. Stevens Jr., "On the encoding problem for DNA computing," in *3rd DIMACS Workshop on DNA Based Computers*, Philadelphia, June 1997, pp. 230–237.

[26] L. Wang, Q. Liu, A. Frutos, S. Gillmor, A. Thiel, T. Strother, A. Condon, R. Corn, M. Lagally, and L. Smith, "Surface-based DNA computing operations: Destroy and readout," in *Fourth International Meeting on DNA Based Computers*, Philadelphia, June 1998, pp. 247–248.

[27] M. Amos and P. Dunne, "DNA simulation of Boolean circuit," Department of Computer Science, University of Warwick, UK, 1998, http://www.csc.liv.ac.uk/~ctag/archive/t/CTAG-97009.ps.

[28] R. Beigel and B. Fu, "Molecular approximation algorithm for NP optimization problems," in *3rd DIMACS Workshop on DNA Based Computers*, Philadelphia, June 1997, pp. 93–104.

[29] A. Gehani and J. Reif, "Micro flow bio-molecular computation," in *Fourth International Meeting on DNA Based Computers*, Philadelphia, June 1998, pp. 253–266.

[30] N. Jonoska and S A. Karl, "A molecular computation of the road coloring problem," in *Second Annual Meeting on DNA Based Computers*, Princeton, June 1996, pp. 148–158.

[31] K U. Mir, "A restricted genetic alphabet for DNA computing," in *Second Annual Meeting on DNA Based Computers*, Princeton, June 1996, pp. 128–130.

[32] L. Adleman, P W. K. Rothemund, S. Roweis, and E. Winfree, "On applying molecular computation to the data encryption standard," in *Second Annual Meeting on DNA Based Computers*, Princeton, June 1996, pp. 28–48.

[33] R S. Braich, C. Johnson, P W. K. Rothemund, D. Hwang, N. Chelyapov, and L. Adleman, "Solution of a satisfiability problem on a gel-based DNA computer," in *Sixth International Meeting on DNA Based Computers*, Leiden, Netherland, June 2000, pp. 31–42.

[34] S. Roweis, E. Winfree, R. Burgoyne, N. Chelyapov, M. Goodman, P. Rothemund, and L. Adleman, "A sticker based architecture for DNA computation," in *Second Annual Meeting on DNA Based Computers*, Princeton, June 1996, pp. 1–27.

[35] S. Roweis, E. Winfree, R. Burgoyne, N. Chelyapov, M. Goodman, P. Rothemund, and L. Adleman, "A sticker based architecture for DNA computation," in *Journal of Computational Biology*, Larchmont, 1998.

[36] N. Seeman, F. Liu, C. Mao, X. Yang, L. Wenzler, and E. Winfree, "DNA nanotechnology: Control of 1-d and 2-d arrays and the construction of a nanomechanical device," in *Fourth International Meeting on DNA Based Computers*, Philadelphia, June 1998, pp. 241–244.

[37] E. Winfree, X. Yang, and N. Seeman, "Universal computation via self-assembly of DNA: Some theory and experiments," in *Second Annual Meeting on DNA Based Computers*, Princeton, June 1996, pp. 172–190.

[38] E. Winfree, "Proposed techniques," in *Fourth International Meeting on DNA Based Computers*, Philadelphia, June 1998, pp. 175–188.

[39] E. Winfree, "Simulations of computing by self-assembly," in *Fourth International Meeting on DNA Based Computers*, Philadelphia, June 1998, pp. 213–240.

[40] E. Winfree, "Algorithmic self-assembly of DNA," Ph.D. dissertation, California Institute of Technology, Pasadena, June 1998.

[41] J. Reif, "Local parallel biomolecular computation," in *3rd DIMACS Workshop on DNA Based Computers*, Philadelphia, June 1997, pp. 243–264.

[42] R. Williams and D. Wood, "Exascale computer algebra problems interconnect with molecular reactions and complexity theory," in *Second Annual Meeting on DNA Based Computers*, Princeton, June 1996, pp. 260–268.

[43] A. Hartemink, D. Gifford, and J. Khodor, "Automated constraint-based nucleotide sequence selection for DNA computation," in *Fourth International Meeting on DNA Based Computers*, Philadelphia, June 1998, pp. 287–297.

[44] J. Khodor and D. Gifford, "Design and implementation of computational systems based on programmed mutagenesis," in *Fourth International Meeting on DNA Based Computers*, Philadelphia, June 1998, pp. 101–108.

[45] A. Marathe, A. Condon, and R. Corn, "On combinatorial DNA word design," in *Fifth International Meeting on DNA Based Computers*, Cambridge, 1999, pp. 75–88.

[46] M. Hagiya and M. Arita, "Towards parallel evaluation and learning of Boolean mu-formulas with molecules," in *3rd DIMACS Workshop on DNA Based Computers*, Philadelphia, June 1997, pp. 105–114.

[47] M. Ogihara and A. Ray, "Simulating Boolean circuits on a DNA computer," in *First Annual Conference on Computational Molecular Biology*, New York, 1997, pp. 326–331.

[48] M. Ogihara and A. Ray, "DNA-based self-propagating algorithm for solving bounded-fan-in Boolean circuits," in *Third Conference on Genetic Programming*, Philadelphia, 1998, pp. 725–730.

[49] M. Ogihara and A. Ray, "Circuit evaluation: Thoughts on a killer application in DNA computing," in *Computing with Bio-Molecules. Theory and Experiments*, Heidelberg, Germany, 1998, Springer-Verlag.

[50] M. Ogihara, "Relating the minimum model for DNA computation and Boolean circuits," in *Proceedings of the Genetic and Evolutionary Computation Conference*, San Francisco, 1999, pp. 1817–1821.

[51] M. Ogihara and A. Ray, "Executing parallel logical operations with DNA," in *Proceedings of the IEEE Congress on Evolutionary Computation*, Washington, 1999, pp. 972–979.

[52] M. Conrad and K. Zauner, "Design for a DNA conformational processor," in *3rd DIMACS Workshop on DNA Based Computers*, Philadelphia, June 1997, pp. 290–295.

[53] M. Ogihara and A. Ray, "The minimum DNA computation model and its computational power," in *Unconventional Models of Computation*, Heidelberg, Ger-

many, 1998, pp. 309–322.

[54] Z. Qiu and M. Lu, "Take advantage of the computing power of DNA computers," in *Lecture Notes in Computer Science*, Heidelberg, Germany, May 2000, pp. 0570–0577, Springer Verlag Heidelberg.

[55] Z. Qiu and M. Lu, "A surface-based DNA algorithm for the expansion of symbolic determinants," in *Lecture Notes in Computer Science*, Heidelberg, Germany, May 2000, pp. 0653–0659, Springer Verlag Heidelberg.

[56] M. Ogihara and A. Ray, "Breadth first search 3SAT algorithms for DNA computers," in *Technical Report TR629 University of Rochester*, Rochester, 1996.

[57] J. Khodor and D. Gifford, "The efficiency of sequence-specific separation of DNA mixtures for biological computing," in *3rd DIMACS Workshop on DNA Based Computers*, Philadelphia, June 1997, pp. 26–34.

[58] M. Ogihara and A. Ray, "DNA-based parallel computation by 'counting'," in *3rd DIMACS Workshop on DNA Based Computers*, Philadelphia, June 1997, pp. 265–274.

[59] T. Eng, "Linear DNA self-assembly with hairpins generates the equivalent of linear context-free grammars," in *3rd DIMACS Workshop on DNA Based Computers*, Philadelphia, June 1997, pp. 296–304.

[60] M. G. Lagoudakis and T.H. LaBean, "2D DNA self assembly for satisfiability," in *5th DIMACS Workshop on DNA Based Computers*, Cambridge, June 1999.

[61] T. Eng and B. Serridge, "A surface-based DNA algorithm for minimal set cover," in *3rd DIMACS Workshop on DNA Based Computers*, Philadelphia, June 1997, pp. 74–82.

[62] Z. Qiu and M. Lu, "A new approach to advance the DNA computing," in *Journal of Applied Soft Computing*, Orlando, 2003, Elsevier Science.

[63] Z. Qiu and M. Lu, "Dynamic DNA computing model," in *International Journal of Computer Research, special issue on Biocomputing*, Hauppauge, 2003, NovaPublishers.

[64] Z. Qiu and M. Lu, "Efficient DNA computing decoding method," in *Recent Advances in Simulation, Computational Methods and Soft Computing*, Athens, 2002, pp. 240–245, WSEAS Press.

[65] J. Clark and D. Holton, *A First Look at Graph Theory*, World Scientific, River Edge, 1991.

[66] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, 1990.

[67] E. Bach and A. Condon, "DNA models and algorithms for NP-complete problems," *Journal of Computer and System Sciences*, vol. 57, pp. 172–186, 1996.

[68] N. Christofides, *Graph Theory: An Algorithmic Approach*, Academic Press, Orlando, 1975.

[69] P D. Kaplan, G. Cecchi, and A. Libchaber, "DNA-based molecular computation: template-template interactions in PCR," in *Second Annual Meeting on DNA Based Computers*, Princeton, June 1996, pp. 159–171.

[70] D. Boneh and R. Lipton, "A divide and conquer approach to sequencing," in *AMS96*, Augsburg, Germany, 1996.

VITA

Zhiquan Frank Qiu was born in March, 1972 in Harbin, China. He received a B.S. degree in electrical engineering from University of Electronic Science and Technology of China, Chengdu, China, in 1993. He worked as an engineer in Nanjing Electronic Devices Institute, Najing China from 1993-1994. He then received his M.S. degree in electrical and computer engineering from Virginia Polytechnic Institute and State University, Blacksburg, VA, in 2000. He attended Texas A&M University and began working on his Ph.D. degree in computer engineering in the spring of 1997. During the period of Ph.D. study, he was an Assistant Lecturer of the course Microprocessor System Design. He started working in Intel's Wireless Communications & Computing Group October 2001. His major duty at Intel is to design the XScale microprocessor core. His address is 5000 W. Chandler Blvd, Chandler, AZ 85226.

The typist for this thesis was Zhiquan Frank Qiu.